Integer Arithmetic in C

• Addition, subtraction, and multiplication work as you would expect

• Division (/) returns the whole part of the division (the quotient)
  • 12 / 3 is 4
  • 15 / 2 is 7

• Modulus (%) returns the remainder
  • 12 % 3 is 0
  • 15 % 2 is 1
Short hand Operators

- Some operators are shortcuts for others
  - Ex. +=, -=, *=, /=, %=, ++, and --
  - \texttt{x += 5;}
    - is the same as
    - \texttt{x = x + 5;}
  - \texttt{y++;}
    - is the same as
    - \texttt{y = y + 1;}
The Increment (++) and Decrement (--) Operators

• When used by themselves, \( y++ \) and \( ++y \) have identical results
  - In an expression, they have different results

• The relative order of the operator matters:
  - \( y++ \): use \( y \)'s current value, then increment it
  - \( ++y \): increment \( y \), then use the new value

• The same is true for decrement (--)
Operator Precedence

- Precedence rules specify the order in which operators are evaluated
- Remember PMDAS:
  - Parentheses, Multiplication, Division, Addition, Subtraction
- Associativity determines left-right order
Precedence Examples

3 - 8 / 4

/ has the highest precedence, so we compute 8 / 4 first, then subtract the result from 3

Equivalent expression: 3 - (8 / 4)

What is the value of 3 * 4 + 18 / 2?
Precedence Examples

5 - 3 + 4 + 2 - 1 + 7

• + and - have equal precedence, so this expression is evaluated left to right:

( (((((5 - 3) + 4) + 2) - 1) + 7))

• The innermost parentheses are evaluated first
Parentheses

- Parentheses can be used to force a different order of evaluation:
  - $12 - 5 \times 2$ produces 2
  - $(12 - 5) \times 2$ produces 14
Expression Examples

What do the following expressions evaluate to?

1 + 2 * 3

(1 + 2) * 3

13 % 5

23 % 4 * 6
More Expression Examples

- $27.0 / 6.0$
- $27.0 / 6$
- $27 / 6$
- Given:
  
  ```
  int x = 5;
  • int y = x++ * 6;
  • int y = ++x * 6;
  ```
printf() and scanf() revisited

- Each of these functions takes a list of arguments (input values):
  - a control string
  - an optional list of other arguments (data)
- The control string determines how the other arguments are displayed
Control Strings

• A control string may contain one or more conversion specifications (formats)
  
• conversion specifications are replaced (or substituted) by the arguments that follow the control string, in order
  
• They begin with a % and end with a conversion character
  
• For example, the statement

  `printf("%s", "abc");`
  
  will replace “%s” with “abc” in the final output
printf() Conversion Characters

<table>
<thead>
<tr>
<th>Conversion character</th>
<th>How the corresponding argument is printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>as a character</td>
</tr>
<tr>
<td>d</td>
<td>as a decimal integer</td>
</tr>
<tr>
<td>e</td>
<td>as a floating-point number in scientific notation</td>
</tr>
<tr>
<td>f</td>
<td>as a floating-point number</td>
</tr>
<tr>
<td>g</td>
<td>in the e-format or f-format, whichever is shorter</td>
</tr>
<tr>
<td>s</td>
<td>as a string</td>
</tr>
</tbody>
</table>
Three Equivalent Statements

\[
\text{printf(“abc”);}
\]

\[
\text{printf(“%s”, “abc”);}
\]

\[
\text{printf(“%c%c%c”, ‘a’, ‘b’, ‘c’);}
\]
Fields

- A field is the area where an argument is printed.
- The field width is the number of characters that make up the field.
- Field width can be specified as an integer between the % and the conversion character.
- For example,

```c
printf("%c%3c%5c", 'a', 'b', 'c');
```

will print

```
a b c
```
Control Strings for `scanf()`

- `scanf()` is used to collect user input from the keyboard
- It is called with a control string and a list of addresses
- The control string conversion specifiers describe how the input stream characters should be interpreted
- The addresses correspond to the memory locations where variables are stored
Parsing Data

- `scanf()` will skip whitespace (tabs, blanks, and newlines) when reading in numbers.

- Whitespace is **NOT** skipped when `scanf()` is reading in characters.
### scanf() Conversion Characters

<table>
<thead>
<tr>
<th>Conversion character</th>
<th>How input stream characters are converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>as a character</td>
</tr>
<tr>
<td>d</td>
<td>as a decimal integer</td>
</tr>
<tr>
<td>f</td>
<td>as a floating-point number (float)</td>
</tr>
<tr>
<td>If or LF</td>
<td>as a floating-point number (double)</td>
</tr>
<tr>
<td>s</td>
<td>as a string</td>
</tr>
</tbody>
</table>
#include <stdio.h>

int main(void)
{
    char c1, c2, c3;
    int i;
    float x;
    double y;

    printf("\n%s\n%s", "Input three characters,",
           "an int, a float, and a double: ");

    scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);
    printf("\nHere is the data that you typed in:
");
    printf("%3c%3c%3c%5d%17e%17e\n\n",
           c1, c2, c3, i, x, y);
    return 0;
}
Return Values

- `printf()` and `scanf()` each return an integer value when they complete.

- `printf()` returns the number of characters printed, or a negative value if an error occurred.

- `scanf()` returns the number of successful conversions or the system-defined end-of-value.
Flow of Control
Control Flow

- Normally, C programs are executed sequentially.

- We can alter this process using **conditionals** (which provide alternative actions) and **loops** (which repeat groups of statements).
Conditions

- Conditional statements execute a test to determine which path to follow

- This test consists of an expression that is evaluated

- Normally, this expression compares two or more values
True and False Values

• Any expression with a non-zero value is considered to be true
  • Ex. 1, 3.14159, -23

• An expression is only false if its value is 0

• Common programming error: using ‘=’ (assignment) instead of “==” (equality)
  • Ex. if (x = 5)
# Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>age &lt; 30</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>height &gt; 6.2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>taxable &lt;= 20000</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than/equal to</td>
<td>temp &gt;= 98.6</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
<td>grade == 100</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td>number != 250</td>
</tr>
</tbody>
</table>
The `if` Statement

• General form:

```c
if ( condition )
    statement (or block of statements) to be executed if condition is true
```

• Ex.

```c
if ( length < 2 )
    printf("Too short!\n");
```
The if-else Statement

- Select one of two possible execution paths, based on the result of a comparison

General format:

```java
if ( expression )
    statement block 1
else
    statement block 2
```
Compound Statements

- **if** and **else** only execute a single following statement

- We can get around this by enclosing multiple statements in curly braces
  - The resulting block is called a *compound statement*

- Style suggestion: always use curly braces around the body of an **if** or **else** clause
if (key == 'F')
{
    contemp = (5.0/9.0) * (intemp - 32.0);
    printf("Converted to Celsius\n");
}
else
{
    contemp = (9.0/5.0) * intemp + 32.0;
    printf("Converted to Fahrenheit.\n");
}
Iterative Programming

• Many programs perform the same task many times
  • Operations are repeated on different data
• Ex. Adding a list of numbers
• Ex. Displaying a menu of options
• Repetitive tasks are specified using loops
Loop Elements

- All loop constructs share four basic elements:
  1. Initialization
  2. Testing the loop condition
  3. The loop body (the task to be repeated)
  4. The loop update

- The order of these elements may vary
Initialization

• This section of code is used to set starting values

• For example, setting a total to 0 initially

• This can be done as part of the loop, or separately before the loop code begins
Loop Tests

- Test expressions are used to determine whether the loop should execute (again)

- Tests compare one value/variable with another

- If the test evaluates to TRUE, then the loop will execute another time
Loop Update

• This step changes the value(s) of the loop variable(s) before the loop repeats

• Ex. moving to the next item to process

• This can be done explicitly as part of the loop, or it can be done inside the loop body
while Loops

- **while** loops can execute an arbitrary number of times
- Order of execution:
  1. Initialization
  2. Loop condition test
  3. Loop body
  4. Loop update
General Form

initialization

while ( loop condition test )
{
    loop body
    loop update
}

**while Loop Example**

```c
int countDown = 5;
while (countDown >= 0)
{
    printf("%d...", countDown);
    countDown--;
}
```
Loop Output

5...4...3...2...1...0...
Another Example

```c
int root = 0;
while (root < 10)
{
    root += 1;
    printf("%d * %d = ", root, root);
    printf("%d\n", root * root);
}
```
<table>
<thead>
<tr>
<th>root</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 * 1 = 1</td>
</tr>
<tr>
<td>1</td>
<td>2 * 2 = 4</td>
</tr>
<tr>
<td>2</td>
<td>3 * 3 = 9</td>
</tr>
<tr>
<td>3</td>
<td>4 * 4 = 16</td>
</tr>
<tr>
<td>4</td>
<td>5 * 5 = 25</td>
</tr>
<tr>
<td>5</td>
<td>6 * 6 = 36</td>
</tr>
<tr>
<td>6</td>
<td>7 * 7 = 49</td>
</tr>
<tr>
<td>7</td>
<td>8 * 8 = 64</td>
</tr>
<tr>
<td>8</td>
<td>9 * 9 = 81</td>
</tr>
<tr>
<td>9</td>
<td>10 * 10 = 100</td>
</tr>
</tbody>
</table>
for Loops

- **for** loops execute a fixed number of times

- Order of execution:
  1. Initialization
  2. Loop condition test
  3. Loop body
  4. Loop update
General Form

\[
\text{for ( initialization ; } \\
\text{\quad loop condition test ; } \\
\text{\quad loop update )} \\
\{
\text{loop body}
\}
\]
for Loop Example

```c
int i;
for (i = 0; i < 10; i++)
{
    printf("%d ", i);
}
```
Loop Output

0123456789
int nextNumber, i, sum = 0;
for (i = 0; i < 5; i++)
{
    printf("\nEnter a number: ");
    scanf("%d ", nextNumber);
    sum += nextNumber;
}
<table>
<thead>
<tr>
<th>i</th>
<th>nextNumber</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>32</td>
</tr>
</tbody>
</table>