Reading Assignment #3

Topic: C

Textbook: Problem Solving & Program Design in C
by Jeri R. Hanly and Elliot B. Koffman

Chapter (2)

Overview of C – Variables & Data

Types, arithmetic expressions,
formatting of program output,
general form of C programs.
You enter the program and save it as a source file

Source file on disk

The compiler attempts to translate the program

Revised source file

You correct syntax errors

Success

Failure

List of errors

New object file

Other object files

The linker links the new object file with other object files

Load file

The loader places the load file into memory

Executable program in memory

Preparing a Program for Execution
Flow of Information during Program Execution

Step 1
Input data: meter readings
Program input

Step 2
Memory
Machine language program for computing water bill
Data entered during execution
Computed results
Program output
Central processor unit

Step 3
Output results: water bill
Miles-to-Kilometers Conversion Program

/*
 * Converts distances from miles to kilometers.
 */

#include <stdio.h>    /* printf, scanf definitions */
#define KMS_PER_MILE 1.609 /* conversion constant */

int
main(void)
{
    double miles, /* distance in miles */
      kms; /* equivalent distance in kilometers */

    /* Get the distance in miles. */
    printf("Enter the distance in miles> ");
    scanf("%lf", &miles);

    /* Convert the distance to kilometers. */
    kms = KMS_PER_MILE * miles;

    /* Display the distance in kilometers. */
    printf("That equals %f kilometers.\n", kms);

    return (0);
}

Enter the distance in miles> 112.0
That equals 180.208000 kilometers.
Declaring Variables in C

4 Variable Types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer</td>
<td>7, 4, 1000</td>
</tr>
<tr>
<td>double</td>
<td>real numbers with double precision</td>
<td>7.4, 4.1, 1000.6</td>
</tr>
<tr>
<td>char</td>
<td>characters ‘d’, ‘B’, ‘o’</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>real numbers with single precision</td>
<td>3 1/3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.3333...</td>
</tr>
</tbody>
</table>

```c
double miles;
miles;               // Valid
```

```c
double miles, kms;
kms;                 // Valid
```

```c
kms Not Valid
miles Not Valid
miles/hr Valid
```
**Variable Updating**

```plaintext
double miles;
int kms;
miles = 1.15E-3;
miles = 0.00115;
miles = 1.15e-3;
```

```plaintext
double miles = 1.15E-3;
```

```plaintext
miles = miles + 65;
miles = miles - 65;
```

**Arithmetic:**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>binary addition</td>
</tr>
<tr>
<td>-</td>
<td>binary subtraction</td>
</tr>
<tr>
<td>*</td>
<td>a * b</td>
</tr>
<tr>
<td>/</td>
<td>a / b</td>
</tr>
<tr>
<td>-</td>
<td>- b</td>
</tr>
<tr>
<td>%</td>
<td>remainder in an integer division</td>
</tr>
<tr>
<td>Mathematical Formula</td>
<td>C Expression</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1. $b^2 - 4ac$</td>
<td>$b * b - 4 * a * c$</td>
</tr>
<tr>
<td>2. $a + b - c$</td>
<td>$a + b - c$</td>
</tr>
<tr>
<td>3. $\frac{a + b}{c + d}$</td>
<td>$(a + b) / (c + d)$</td>
</tr>
<tr>
<td>4. $\frac{1}{1 + x^2}$</td>
<td>$1 / (1 + x * x)$</td>
</tr>
<tr>
<td>5. $a \times -(b + c)$</td>
<td>$a * -(b + c)$</td>
</tr>
</tbody>
</table>

The points illustrated are summarized as follows:

- Always specify multiplication explicitly by using the operator * where needed (1).
- Use parentheses when required to control the order of operator evaluation (3, 4).
- Two arithmetic operators can be written in succession if the second is a unary operator (5).
Mathematical Formula C Expression

1. \( b^2 - 4ac \)  
   \(( b * b) - (4 * a * c)\)

2. \( a + b - c \)  
   \( a + b - c \)

3. \( \frac{a + b}{c + d} \)  
   \( \frac{a + b}{c + d} \)

4. \( \frac{1}{1 + x^2} \)  
   \( \frac{1}{1 + x * x} \)

5. \( a \times - (b + c) \)  
   \( a \times - b + c \)  
   \( a(-b) + c = -ab + c \)

   \(-ab - ac\)
**Rules for Evaluation of Arithmetic Expressions**

a. All parenthesized subexpressions must be evaluated separately. Nested parenthesized subexpressions must be evaluated inside out, with the innermost subexpression evaluated first.

b. *The operator precedence rule.* Operators in the same subexpression are evaluated in the following order:

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

c. *The 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*).
Evaluation Tree and Step-by-Step Evaluation for $v = (p_2 - p_1) / (t_2 - t_1)$

\[
\begin{align*}
  v &= (p_2 - p_1) / (t_2 - t_1) \\
  &= \frac{9.0 - 4.5}{60.0 - 0.0} \\
  &= \frac{4.5}{60.0} \\
  &= 0.075
\end{align*}
\]

Evaluation Tree and Step-by-Step Evaluation for $z = (a + b / 2) + w \cdot y$

\[
\begin{align*}
  z &= (a + b / 2) + w \cdot y \\
  &= (3 + 9 / 2) + 2 \cdot (-5) \\
  &= 4 - 10 \\
  &= -6
\end{align*}
\]
Operation

function calls
  unary + unary - unary & unary *
casts
  * / %
binary + binary -
  < > <= >=
  == !=
  &&
  ||
=

Precedence

highest (evaluated first)

lowest (evaluated last)

Precedence of Operations

\[
\text{ans} = 2 + (\text{int}\, \text{ceil}(80.2) \mod 9 > 10)
\]

\[\begin{array}{c}
\hline
\text{81.0} \\
\hline
\text{81} \\
\hline
\text{0} \\
\hline
\text{2} \\
\hline
\text{0} \\
\hline
\end{array}\]

Step-by-Step Evaluation of Expression with Operations
of Many Precedence Levels
3.14159
0.005
12345.
15.0e-6 (value of 15.0 x 10^-6 or 0.000015)
2.345E2 (value of 2.345 x 10^2 or 234.5)
.12e+6 (value of .12 x 10^6 or 120000.)
1.15E-3 (value of 1.15 x 10^-3 or 0.00115)

Constants of Type double (Real Numbers)

3 / 15 = 0   18 / 3 = 6
15 / 3 = 5   16 / -3 varies
16 / 3 = 5   0 / 4 = 0
17 / 3 = 5   4 / 0 is undefined

Results of Integer Division

7 / 2
\[ \begin{array}{c}
\downarrow \\
3 \\
2 \left\lfloor \frac{7}{2} \right\rfloor \\
6 \\
1 \leftarrow 7 \% 2
\end{array} \]

299 / 100
\[ \begin{array}{c}
\downarrow \\
2 \\
100 \left\lfloor \frac{299}{100} \right\rfloor \\
200 \\
99 \leftarrow 299 \% 100
\end{array} \]

Examples of Integer Quotient and Integer Remainder

3 \% 5 = 3   5 \% 3 = 2
4 \% 5 = 4   5 \% 4 = 1
5 \% 5 = 0   15 \% 5 = 0
6 \% 5 = 1   15 \% 6 = 3
7 \% 5 = 2   15 \% -7 varies
8 \% 5 = 3   15 \% 0 is undefined

The \% Operator
# include <stdio.h >

# define PI 3.1415926  → Global constant declaration
       or
double pi = 3.1415926  → Global variable declaration

main ( )
{
    double radius ;
    double area ;
    area = sqr(radius) * pi
}

double sqr( double a )
{
    return ( a * a ) ;
}

int k = 5, m = 4, n;
double x = 1.5, y = 2.1, z;

<table>
<thead>
<tr>
<th>Context of Conversion</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression with binary operator and operands of different numeric types</td>
<td>k + x</td>
<td>Value of int variable k is converted to type double format before operation is performed.</td>
</tr>
<tr>
<td>assignment statement</td>
<td>z = k / m ;</td>
<td>Expression is evaluated first. Then, the result is converted to type double format for assignment.</td>
</tr>
<tr>
<td>assignment statement</td>
<td>n = x * y ;</td>
<td>Expression is evaluated first. Then, the result is converted to type int format for assignment, and fractional part is lost.</td>
</tr>
</tbody>
</table>

Automatic Conversion of Numeric Data
Casting:

Take one type of variable and cast it as another type.

Example:

\[ z = \frac{k}{m}, \text{ we want } z = 1.25 \ (5/4) \]

\[ z = ((\text{double})k)/m \; \rightarrow \; z = 1.25 \]

\[ n = ((\text{int})x) \ast ((\text{int})y) \; \rightarrow \; n = 2 \]
Earlier example:

miles = miles + 65;

equivalent to

miles += 65;

-=

*=

/=
<table>
<thead>
<tr>
<th>Statement with Simple Assignment Operator</th>
<th>Equivalent Statement with Compound Assignment Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ct = ct + 1 ;</code></td>
<td><code>ct += 1 ;</code></td>
</tr>
<tr>
<td><code>time = time - 1 ;</code></td>
<td><code>time -= 1 ;</code></td>
</tr>
<tr>
<td><code>total_pay = total_pay + pay ;</code></td>
<td><code>total_pay += pay ;</code></td>
</tr>
<tr>
<td><code>product = product * data ;</code></td>
<td><code>product *= data ;</code></td>
</tr>
<tr>
<td><code>n = n * (x + 1) ;</code></td>
<td><code>n *= x + 1 ;</code></td>
</tr>
</tbody>
</table>

Using Compound Assignment Operators
Before...

\[
\begin{array}{c}
i \quad j \\
2 \quad \_?
\end{array}
\]

Increments...

\[
j = ++i;
\]

prefix:
Increment 1 and then use it.

\[
j = i++;
\]

postfix:
Use i and then increment it.

After...

\[
\begin{array}{c}
i \quad j \\
3 \quad 3
\end{array}
\]

\[
\begin{array}{c}
i \quad j \\
3 \quad 2
\end{array}
\]

Comparison of Prefix and Postfix Increments
Effect of `scanf("%lf", &miles);`
In general:

```c
scanf ( "%lf" , &miles ) ;
control string variable address

printf ( "%f", miles ) ;
control string variable names
```
To read in other types:

```
%\d    argument is integer
%\c    argument is character
%\s    argument is string
%\lf   argument is double
```

To output (or print) other types:

```
%\d    argument is integer
%\c    argument is character
%\s    argument is string
%\lf   argument is double
```

Example:

double miles;
int year;

scanf(" %lf %d", &miles, &year);

```
miles ----> 30.5
year ----> 1996
```

printf("%f %d", miles, year);

20
<table>
<thead>
<tr>
<th>Value</th>
<th>Placeholder</th>
<th>Printed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>234</td>
<td>%4d</td>
<td>#234</td>
</tr>
<tr>
<td>234</td>
<td>%5d</td>
<td>##234</td>
</tr>
<tr>
<td>234</td>
<td>%6d</td>
<td>###234</td>
</tr>
<tr>
<td>-234</td>
<td>%4d</td>
<td>-234</td>
</tr>
<tr>
<td>-234</td>
<td>%5d</td>
<td>#-234</td>
</tr>
<tr>
<td>-234</td>
<td>%6d</td>
<td>###-234</td>
</tr>
<tr>
<td>234</td>
<td>%2d</td>
<td>234</td>
</tr>
<tr>
<td>234</td>
<td>%1d</td>
<td>234</td>
</tr>
<tr>
<td>-234</td>
<td>%2d</td>
<td>-234</td>
</tr>
</tbody>
</table>

Printing 234 and -234 Using Different Placeholders

<table>
<thead>
<tr>
<th>x</th>
<th>Output Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>-99.42</td>
<td>-99.42</td>
</tr>
<tr>
<td>0.123</td>
<td>##0.12</td>
</tr>
<tr>
<td>-9.536</td>
<td>#-9.54</td>
</tr>
<tr>
<td>-25.554</td>
<td>-25.55</td>
</tr>
<tr>
<td>99.999</td>
<td>100.00</td>
</tr>
<tr>
<td>999.4</td>
<td>999.40</td>
</tr>
</tbody>
</table>

Displaying x Using format String Placeholder %6.2f

<table>
<thead>
<tr>
<th>Value</th>
<th>Placeholder</th>
<th>Printed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14159</td>
<td>%5.2f</td>
<td>#3.14</td>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<td>3.14159</td>
<td>%8.5f</td>
<td>#3.14159</td>
</tr>
<tr>
<td>3.14159</td>
<td>%8.4f</td>
<td>3.1416</td>
</tr>
<tr>
<td>0.1234</td>
<td>%4.2f</td>
<td>0.12</td>
</tr>
<tr>
<td>-0.006</td>
<td>%4.2f</td>
<td>-0.01</td>
</tr>
<tr>
<td>-0.006</td>
<td>%8.4f</td>
<td>-0.0060</td>
</tr>
<tr>
<td>-0.006</td>
<td>%8.5f</td>
<td>-0.00600</td>
</tr>
<tr>
<td>-0.006</td>
<td>%8.3f</td>
<td>##-0.006</td>
</tr>
</tbody>
</table>

Formatting Values of Type double
\n : print a new line
\t : print a tab
\" : print "
\\ : print a \n
That equals 180.208000 kilometers.
That equals 180.20800 kilometers.
<\tab>
Let’s write a simple program to read in the three letter name for a person, and also the current year, and print the following message:
"Welcome Bob, 1999 is a great year"

Answer

```c
#include <stdio.h>
main (void)
{
char letter_1, letter_2, letter_3; /* the three letter name */
int year; /* the current year */
printf("Enter three letter nickname and press return :");
scanf("%c %c %c", &letter_1, &letter_2, &letter_3);
printf("Enter the current year and press return :");
scanf("%d", &year);
printf("Welcome %c%c%c, %d is a great year\n", letter_1, letter_2, letter_3, year);
return (0);
}
```

Example Execution

Enter a three letter nickname and … > Bob
Enter the current year … >1999
Welcome Bob, 1999 is a great year
Effect of `scanf("%c%c%c", &letter_1, &letter_2, &letter_3);

Welcome, %c%c%c.  %d is a great ...

Substitution of Print List Values for Placeholders
/*
 * Incorrect revision of welcoming message program
 */
#include <stdio.h> /* printf, scanf definitions */

int
main(void)
{
    char letter_1, letter_2, letter_3; /* three letters */
    int year; /* current year */

    printf("Enter the current year and press return> ");
    scanf("%d", &year);
    printf("Enter a 3-letter nickname and press return> ");
    scanf("%c%c%c", &letter_1, &letter_2, &letter_3);
    printf("Welcome, %c%c%c. %d is a great year to study C!\n", 
            letter_1, letter_2, letter_3, year);

    return (0);
}

Enter the current year and press return> 1993
Enter a 3 letter nickname and press return> Bob
Welcome, 
Bo. 1993 is a great year to study C!

Program with Incorrect Results Due to Character
and Numeric Scanning Problem

letter_1  \n
letter_2  B

letter_3  0

year  1993

Memory Contents during Execution of
Faulty Welcome Message Program