Computer Programming Using C
COP 3275 - Summer 2017

Lecture 5: C Fundamentals (cont.)

By Ahmed E. Khaled

/* Programming */
Recap to previous lecture!

- Printing the value of a variable (printf)
- Taking an input from user (scanf)
Defining Names for Constants

• Constant is a value that can not be changed during the execution.

• Using a feature known as *macro definition*, we can name this constant:

```
#define MY_CONSTANT 166
```
Defining Names for Constants

• When a program is compiled, the *preprocessor* replaces each macro by the value that it represents.

• During preprocessing, the statement

weight = volume / MY_CONSTANT;

will become

weight = volume / 166;
Defining Names for Constants

• The value of a macro can be an expression:
  
  ```
  #define RECIPROCAL_OF_PI (1.0 / 3.14159)
  ```

• If it contains operators, the expression should be enclosed in parentheses.

• Using only upper-case letters in macro names is a common convention (not a language requirement).
Program: Converting from Fahrenheit to Celsius

- The `celsius.c` program prompts the user to enter a Fahrenheit temperature; it then prints the equivalent *Celsius temperature*.

- Sample program output:
  
  Enter Fahrenheit temperature: 212
  Celsius equivalent: 100.0

- The conversion formulas between °F and °C are:
  
  \[ [\degree C] = ( [\degree F] - 32 ) \times \frac{5}{9} \]
celsius.c

/* Converts a Fahrenheit to Celsius */

#include <stdio.h>

#define F_FACTOR 32.0
#define S_FACTOR (5.0 / 9.0)

main()
{
    float fahrenheit, celsius;

    printf("Enter Fahrenheit temperature: ");
    scanf("%f", &fahrenheit);

    celsius = (fahrenheit - F_FACTOR) * S_FACTOR;

    printf("Celsius equivalent: %.1f\n", celsius);
}
Identifiers

• Identifier is the name for a variable, function, constant and any other entity in your program.

• C places *no limit* on the maximum length of an identifier.

• An identifier in C may contain letters, digits, and underscores, *but must begin with a letter or underscore*: Correct identifiers: `times10` `get_char_N` `_done`
Incorrect identifiers: `10times` `get-next-char`
Identifiers

• C is *case-sensitive*, the following identifiers are all different:

  job  joB  jOb  jOB  Job  JoB  JOb  JOB
The following *keywords* can’t be used as identifiers:

<table>
<thead>
<tr>
<th>auto</th>
<th>enum</th>
<th>restrict</th>
<th>unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>extern</td>
<td>return</td>
<td>void</td>
</tr>
<tr>
<td>case</td>
<td>float</td>
<td>short</td>
<td>volatile</td>
</tr>
<tr>
<td>char</td>
<td>for</td>
<td>signed</td>
<td>while</td>
</tr>
<tr>
<td>const</td>
<td>goto</td>
<td>sizeof</td>
<td>_Bool</td>
</tr>
<tr>
<td>continue</td>
<td>if</td>
<td>static</td>
<td>_Complex</td>
</tr>
<tr>
<td>default</td>
<td>inline</td>
<td>struct</td>
<td>_Imaginary</td>
</tr>
<tr>
<td>do</td>
<td>int</td>
<td>switch</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>long</td>
<td>typedef</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td>register</td>
<td>union</td>
<td></td>
</tr>
</tbody>
</table>
Expressions in $C$
Formulas to compute a value
Expressions

• Operators are the basic tools for building an expression in C.

• Three types of operators exist in C:
  – *Arithmetic operators*: $a + b$
  – *Relational operators* (comparisons): $a > b$
  – *Logical operators* (condition): $a > 0$ and $a < 100$
Arithmetic Operators

C provides two unary and five binary arithmetic operators:

– unary: Unary plus and Unary minus
– binary: Additive and Multiplicative

1. Additive: +, -
2. Multiplicative: x, / and %
Unary operators

#include <stdio.h>
main()
{
    int x = 3;
    printf("Value of X is %d \n",x);

    x = +1;
    printf("Value of X is %d \n",x);

    int i = 5;
    x = -i;
    printf("Value of X is %d \n",x);
}

Binary Operators

#include <stdio.h>

main()
{
    int i = 5;
    int j = i;
    int x = 10 * i + j;
    printf("Value of X is %d \n",x);
}
Operator Precedence and Associativity

- When an expression contains more than one operator, its interpretation depends on operator precedence rules.

- For example, what is the value of $l + j * K$?
Operator precedence rules

Highest: + - (unary)

* / %

Lowest: + - (binary)

• $I + j * k$  ->  $I + (j * k)$
• $-I * -j$  ->  $(-I) * (-j)$
• $+I + j / k$  ->  $(+1) + (j / k)$
When the expression contains two or more operators of the same level of precedence, the compiler applies the *left associativity*.

- $I - j - k \Rightarrow (I - j) - k$
- $I \times j / k \Rightarrow (I \times j) / k$
Special Cases for / and %

When both operands of / are integers, the result is truncated.

```c
#include <stdio.h>

main()
{
    int y = 1/2;
    printf("Value of X is %d \n", y); //0

    float x = 1/3;
    printf("Value of X is %f \n", x); //0.0

    x = 1.0/3.0;
    printf("Value of X is %f \n", x); //0.3
}
```
Special Cases for / and %

When any operand of % is not integer, the program won’t compile.

```
#include <stdio.h>
main()
{
    int y = 34%10;
    printf("Value of Y is %d \n",y); //4

    float x = 34%10;
    printf("Value of X is %f \n",x); //4.0
}
```
Compound Assignment

• Assignments that use the *old value* of a variable to compute its *new value*.

• Example:

```plaintext
int i = 4;
i = i + 2;
i = i + 2;
```
Compound Assignment

- Some programmers use the compound assignment operator as follows:

```c
i += 2;
/* same as i = i + 2; */
```
Compound Assignment

• There are nine other compound assignment operators, including the following:

- \( v -= e \)
- \( v *= e \)
- \( v /= e \)
- \( v %= e \)

• All compound assignment operators work in much the same way:

\( v += e \) adds \( v \) to \( e \), storing the result in \( v \)
\( v -= e \) subtracts \( e \) from \( v \), storing the result in \( v \)
\( v *= e \) multiplies \( v \) by \( e \), storing the result in \( v \)
\( v /= e \) divides \( v \) by \( e \), storing the result in \( v \)
Increment and Decrement Operators

- C provides special `++` (increment) and `--` (decrement) operators.
- The `++` operator adds 1 to its operand. The `--` operator subtracts 1.

```
i = i + 1;     i += 1;     i++;  
j = j - 1;     j -= 1;     j--;  
```
Exercises

- Write a program that asks the user to enter a two-digit number, and print the number with its digits reversed.

Enter a two-digit number: 28
The reversal is: 82
#include <stdio.h>

main()
{
    int value;
    printf("Enter a two-digit number: ");
    scanf("%d", &value);

    int a = value / 10;
    int b = value % 10;

    int result = (b * 10) + a;
    printf("The reversal is: %d \n", result);
}
Extend the program to handle *three-digit* number!