**printf()**

Standard Library Function

- Used to write text to the "standard output"
- Normally a computer monitor or printer
- Often the UART in embedded systems
- SIM Uart1 window in MPLAB-SIM

### Syntax

```c
printf(ControlString, arg1,...argn);
```

- Everything printed verbatim within string except %d's which are replaced by the argument values from the list

### Example

```c
int a = 5, b = 10;
printf("a = %d\n b = %d\n", a, b);
```

**Result:**

```
a = 5
b = 10
```

**NOTE:** the 'd' in %d is the conversion character.
(See next slide for details)
printf() Conversion Characters for Control String

<table>
<thead>
<tr>
<th>Conversion Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%c</td>
<td>Single character</td>
</tr>
<tr>
<td>%s</td>
<td>String (all characters until '\0')</td>
</tr>
<tr>
<td>%d</td>
<td>Signed decimal integer</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal integer</td>
</tr>
<tr>
<td>%o</td>
<td>Unsigned octal integer</td>
</tr>
<tr>
<td>%x</td>
<td>Unsigned hexadecimal integer with lowercase digits (\x5e)</td>
</tr>
<tr>
<td>%X</td>
<td>As x, but with uppercase digits (e.g. \x5E)</td>
</tr>
<tr>
<td>%f</td>
<td>Signed decimal value (floating point)</td>
</tr>
<tr>
<td>%e</td>
<td>Signed decimal with exponent (e.g. 1.26e-5)</td>
</tr>
<tr>
<td>%E</td>
<td>As e, but uses E for exponent (e.g. 1.26E-5)</td>
</tr>
<tr>
<td>%g</td>
<td>As e or f, but depends on size and precision of value</td>
</tr>
<tr>
<td>%G</td>
<td>As g, but uses E for exponent</td>
</tr>
</tbody>
</table>

printf() Gotchas

- The value displayed is interpreted entirely by the formatting string:
  
  ```
  printf("ASCII = \%d", 'a');
  ```

  will output: ASCII = 97

  A more problematic string:
  
  ```
  printf("Value = \%d", 6.02e23);
  ```

  will output: Value = 26366

- Incorrect results may be displayed if the format type doesn't match the actual data type of the argument
printf() Useful Format String Examples for Debugging

• Print a 16-bit hexadecimal value with a "0x" prefix and leading zeros if necessary to fill a 4 hex digit value:

```c
printf("Address of x = %#06x\n", x_ptr);
```

# Specifies that a 0x or 0X should precede a hexadecimal value (has other meanings for different conversion characters)

06 Specifies that 6 characters must be output (including 0x prefix), zeros will be filled in at left if necessary

x Specifies that the output value should be expressed as a hexadecimal integer

printf("Address of x = 0x%04X\n", x_ptr);

04 Specifies that 4 characters must be output (no longer including 0x prefix since that is explicitly included in the string), zeros will be filled in at left if necessary

X Specifies that the output value should be expressed as a hexadecimal integer with uppercase A-F
Lab Exercise 3

printf() Library Function

Exercise 03
printf() Library Function

• Open the lab Project:

  On the class website
  Examples -> Lab3.zip

  Open MPLAB®X and select Open Project Icon (Ctrl + Shift + O)
  Open the Project listed above.

  If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”
Exercise 03
printf() Library Function

• Compile and run the code:

2. Click on the Debug Project button.

3. If no errors are reported, click on Continue button to start the program.

4. Wait for the UART1 Output to finish the click on the Pause button.
Exercise 03
printf() Library Function

• Expected Results (1):

The UART1 Output window should show the text that is output by the program by printf(), showing the how values are printed based on the formatting character used in the control string.

Lab 03
printf() Library Function

• Expected Results (2):

<table>
<thead>
<tr>
<th>Detailed Analysis:</th>
<th>Line of Code From Demo Project</th>
<th>Output</th>
</tr>
</thead>
</table>
| printf(“25 as decimal (d): %d
”, 25);      |                                                        | 25              |
| printf(“’a’ as character (c): %c
”, ’a’);  |                                                        | a               |
| printf(“’a’ as decimal (d): %d
”, ’a’);   |                                                        | 97              |
| printf(“2.55 as float (f): %f
”, 2.55);   |                                                        | 2.550000        |
| printf(“2.55 as decimal (d): %d
”, 2.55); |                                                        | 16419           |
| printf(“6.02e23 as exponent (e): %e
”, 6.02e23); |                                                      | 6.020000e+23   |
| printf(“6.02e23 as decimal (d): %d
”, 6.02e23); |                                                      | 26366           |
| printf(“’Microchip’ as string (s): %s
”, ”Microchip”); |                                        | Microchip       |
| printf(“’Microchip’ as decimal (d): %d
”, ”Microchip”); |                                                       | 24058           |
Exercise 03

Conclusions

• printf() has limited use in embedded applications themselves
• It is very useful as a debugging tool
• It can display data almost any way you want
• Projects that use printf() must:
  – Configure a heap (done in MPLAB®X -IDE)
  – Include the stdio.h header file

scanf() 
Standard Library Function

• Used to read input from the "standard input"
• Normally a keyboard or file
• Often the UART in embedded systems
• Input file into the SIM in MPLAB-X
• sscanf() inputs from string
• fscanf() inputs from file
**scanf()**

Standard Library Function

```c
int scanf(FormatString, arg1,...argn);
```

- The format string tells `scanf` what kind of input.
- `arg1` through `argn` are **POINTERS** to variable of the right type.

**Example**

```c
int a, b;
printf("Input a and b\n");
scanf("%d", &a, &b);
printf("a=%d,b=%d",a,b);
```

```
Result:
a = 5
b = 10
```

---

**scanf()**

Conversion Characters for Control String

<table>
<thead>
<tr>
<th>Conversion Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>Decimal integer</td>
</tr>
<tr>
<td>%i</td>
<td>Integer, may be in octal (leading 0) or hexadecimal (leading 0x,0X)</td>
</tr>
<tr>
<td>%o</td>
<td>Octal integer, with or without leading 0</td>
</tr>
<tr>
<td>%u</td>
<td>Unsigned decimal integer</td>
</tr>
<tr>
<td>%x</td>
<td>Hexadecimal integer, with or without leading 0x or 0X</td>
</tr>
<tr>
<td>%c</td>
<td>Character, does not skip white space</td>
</tr>
<tr>
<td>%s</td>
<td>Next non-whitespace character</td>
</tr>
<tr>
<td>%s</td>
<td>Character string (not quoted), terminating \0 will be added</td>
</tr>
<tr>
<td>%e,%f,%g</td>
<td>Floating point</td>
</tr>
<tr>
<td>%</td>
<td>Literal “%” sign, no assignment is made</td>
</tr>
<tr>
<td>h</td>
<td>Before d,i,o,u, or x indicates pointer is “short” version</td>
</tr>
<tr>
<td>l</td>
<td>Before d,i,o,u, or x indicates pointer is “long” version</td>
</tr>
</tbody>
</table>
**scanf()**

Gotchas

- Ignores blanks and tabs in format string
- Skips over white space (blanks, tabs, newline) as it looks for input
- Returns number of successful conversions

- Arguments **must** be pointers to variable types

```
("%c %c", Q,a)
```

**scanf()**

Examples

- Read input line with date in the format:
  
  \[ .25 \text{ Dec} \ 2012 \]

  ```
  \text{scanf("\%d \%s \%d", \&day, month, \&year);}\
  ```

  - **day** int, &day is pointer to day
  - **month** char[20], is a string for putting the month into, does not need “&” because name of array is already a pointer
  - **year** int, &year is pointer to year
**scanf()**

Examples

- Read input line with date in the format:
  - 25/12/2012

```c
scanf("%d/%d/%d", &day, &month, &year);
```

- **day** int, &day is pointer to day
- **month** int, &month is pointer to month
- **year** int, &year is pointer to year
```c
for (int i = 0; i < NUM200; i++) {
}
}

for (i = 0; i < NUM200; i++) {
}

5

CMPE-013/L
Expressions and Control
Gabriel Hugh Elkaim
Spring 2012
Expressions

- Represents a single data item (e.g. character, number, etc.)
- May consist of:
  - A single entity (a constant, variable, etc.)
  - A combination of entities connected by operators (+, -, *, / and so on)

Example Expressions

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a + b</code></td>
</tr>
<tr>
<td><code>x == y</code></td>
</tr>
<tr>
<td><code>speed = dist / time</code></td>
</tr>
<tr>
<td><code>z = ReadInput()</code></td>
</tr>
<tr>
<td><code>c &lt;= 7</code></td>
</tr>
<tr>
<td><code>x &gt;= 25</code></td>
</tr>
<tr>
<td><code>count++</code></td>
</tr>
<tr>
<td><code>d = a + 5</code></td>
</tr>
</tbody>
</table>
Statements

- Cause an action to be carried out
- Three kinds of statements in C:
  - Expression Statements
  - Compound Statements
  - Control Statements

Examples

Expression Statements

- An expression followed by a semi-colon
- Execution of the statement causes the expression to be evaluated
Compound Statements

- A group of individual statements enclosed within a pair of curly braces { and }
- Individual statements within may be any statement type, including compound
- Allows statements to be embedded within other statements
- Does NOT end with a semicolon after }
- Also called Block Statements

Example

```c
{ float start, finish;
  start = 0.0;
  finish = 400.0;
  distance = finish - start;
  time = 55.2;
  speed = distance / time;
  printf("Speed = %f m/s", speed);
}
```
Control Statements

- Used for loops, branches and logical tests
- Often require other statements embedded within them

Example

```c
while (distance < 400.0)
{
    printf("Keep running!");
    distance += 0.1;
}
```

(while syntax: `while expr statement`)

Decisions and Branching
Boolean Expressions

• C has no Boolean data type
• Boolean expressions return integers:
  – 0 if expression evaluates as FALSE
  – non-zero if expression evaluates as TRUE (usually returns 1, but this is not guaranteed)

```c
int main(void)
{
    int x = 5, y, z;
    y = (x > 4);  // y = 1 (TRUE)
    z = (x > 6);  // z = 0 (FALSE)
    while (1);
}
```

Equivalent Expressions

• If a variable, constant or function call is used alone as the conditional expression:
  ```c
  (MyVar) or (Foo())
  ```
  This is the same as saying:
  ```c
  (MyVar != 0) or (Foo() != 0)
  ```
• In either case, if MyVar ≠ 0 or Foo() ≠ 0, then the expression evaluates as TRUE (non-zero)
• C Programmers almost always use the first method (laziness always wins in C)
**if Statement**

**Syntax**

`if (expression) statement`

- `expression` is evaluated for boolean TRUE (≠0) or FALSE (=0)
- If TRUE, then `statement` is executed

**Note**

Whenever you see `statement` in a syntax guide, it may be replaced by a compound (block) statement.

Remember: spaces and new lines are not significant.

```
if (expression)
{
  statement1
  statement2
}
```
**if Statement**

Example

```c
{  int x = 5; // a  \( \infty \)  
    if (x) // If x is TRUE (non-zero)...
    {  printf("x = %d\n", x); // ...then print the value of x. 
      // while (1); 
    }
}
```

- What will print if \( x = 5 \)? ... if \( x = 0 \)?
- ...if \( x = -82 \)?
  - **if \( x = 65536 \)?**

**if Statement**

Solution to Trick Question

- If \( x = 65536 \), this is the same as \( x = 0 \)
- Why?
  - An integer, whether signed or unsigned can only hold 16-bit values (65536 requires 17 bits)
  - signed int: -32768 to 32767 (twos complement)
  - unsigned int: 0 to 65535 = \( 2^{16} - 1 \)
**if Statement**

*Testing for TRUE*

- **if (x)** vs. **if (x == 1)**
  - **if (x)** only needs to test for not equal to 0
  - **if (x == 1)** needs to test for equality with 1
  - Remember: TRUE is defined as non-zero, FALSE is defined as zero

---

**Example: if (x)**

```
if (x)
```

| 8: | 011B4 | E208C2 | cp0.w 0x08c2 |
| 10: | 011B6 | 320004 | bra z, 0x0011c0 |

**Example: if (x == 1)**

```
if (x == 1)
```

| 11: | 011C0 | 804610 | mov.w 0x08c2,0x0000 |
| 12: | 011C2 | 500FE1 | sub.w 0x0000,#1,[0x001e] |
| 13: | 011C4 | 3A0004 | bra nz, 0x0011ce |

---

**Nested if Statements**

```
int power = 10;
float band = 2.0;
float frequency = 146.52;

if (power > 5)
{
    if (band == 2.0)
    {
        if ((frequency > 144) && (frequency < 148))
        {
            printf("Yes, it's all true!\n");
        }
    }
}
```
### if-else Statement

**Syntax**

\[
\text{if } (\text{expression}) \ \text{statement}_1 \\
\text{else } \ \text{statement}_2
\]

- expression is evaluated for boolean TRUE (≠0)
  or FALSE (=0)  
- If TRUE, then statement\(_1\) is executed  
- If FALSE, then statement\(_2\) is executed
**if-else Statement**

Example

```c
float frequency = 146.52; // frequency in MHz
if ((frequency > 144.0) && (frequency < 148.0))
{
  printf("You're on the 2 meter band\n");
}
else
{
  printf("You're not on the 2 meter band\n");
}
```

**if-else if Statement**

Syntax

```c
if (expression₁) statement₁
else if (expression₂) statement₂
else statement₃
```

- `expression₁` is evaluated for boolean TRUE (≠0) or FALSE (=0)
- If TRUE, then `statement₁` is executed
- If FALSE, then `expression₂` is evaluated
- If TRUE, then `statement₂` is executed
- If FALSE, then `statement₃` is executed
**if-else if Statement**

**Syntax**

```
if (expression₁) statement₁
else if (expression₂) statement₂
else statement₃
```

**Flow Diagram**

```
START
expression
TRUE
statement₂
FALSE
expression
TRUE
statement₃
FALSE
statement₃
END
```

**Example**

```
if ((freq > 144) && (freq < 148))
    printf("You're on the 2 meter band\n");
else if ((freq > 222) && (freq < 225))
    printf("You're on the 1.25 meter band\n");
else if ((freq > 420) && (freq < 450))
    printf("You're on the 70 centimeter band\n");
else
    printf("You're somewhere else\n");
```
Lab Exercise 5

Making Decisions: if Statements

Exercise 05
Making Decisions (if)

• Open the lab Project:

On the class website
Examples -> Lab5.zip

1 Open MPLAB®X and select Open Project Icon (Ctrl + Shift + O)
Open the Project listed above.

If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”
Exercise 05
Making Decisions (if)

Solution: Steps 1 and 2

```c
/*###########################################################################
# STEP 1: Increment intVariable1 if BOTH the following conditions are true:
#         * floatVariable2 is greater than or equal to floatVariable1
#         * charVariable2 is greater than or equal to charVariable1
#         Remember to use parentheses to group logical operations.
###########################################################################*/
//Write the if condition
if((floatVariable2 >= floatVariable1) && (charVariable2 >= charVariable1))
{
    intVariable1++; //Increment intVariable1
}

/*###########################################################################
# STEP 2: If the above is not true, and floatVariable1 is greater than 50
#         then decrement intVariable2.  (HINT: else if)
###########################################################################*/
//Write the else if condition
else if(floatVariable1 > 50)
{
    intVariable2--; //Decrement intVariable2
}
```
Exercise 05
Making Decisions (if)

Solution: Step 3

```c
/*###########################################################################
# STEP 3: If neither of the above are true, set charVariable2 equal to 1. #
# (HINT: else)
###########################################################################*/

//Write the else condition
else
{
    charVariable2 = 1;  //Set charVariable2 equal to 1
}
```

Exercise 05
Conclusions

- if statements make it possible to conditionally execute a line or block of code based on a logic equation
- else if / else statements make it possible to present follow-up conditions if the first one proves to be false
**switch Statement**

Syntax:
```
switch (expression)
{
    case const-expr: statements
    ...
    case const-expr: statements
    default: statements
}
```

- `expression` is evaluated and tested for a match with the `const-expr` in each `case` clause.
- The `statements` in the matching `case` clause is executed.

**Flow Diagram (default)**

Notice that each statement falls through to the next.

This is the default behavior of the `switch` statement.
switch Statement

Flow Diagram (modified)

Adding a break statement to each statement block will eliminate fall through, allowing only one case clause's statement block to be executed.

switch Example 1

```c
switch (channel) {
    case 2:  printf("WBBM Chicago\n"); break;
    case 3:  printf("DVD Player\n"); break;
    case 4:  printf("WTMJ Milwaukee\n"); break;
    case 5:  printf("WMAQ Chicago\n"); break;
    case 6:  printf("WITI Milwaukee\n"); break;
    case 7:  printf("WLS Chicago\n"); break;
    case 9:  printf("WGN Chicago\n"); break;
    case 10: printf("WMVS Milwaukee\n"); break;
    case 11: printf("WTTW Chicago\n"); break;
    case 12: printf("WISN Milwaukee\n"); break;
    default: printf("No Signal Available\n");
}
```
**switch Statement**

**switch Example 2**

```c
switch(letter) {
  case 'a':
    printf("Letter 'a' found.\n");
    break;
  case 'b':
    printf("Letter 'b' found.\n");
    break;
  case 'c':
    printf("Letter 'c' found.\n");
    break;
  default:  printf("Letter not in list.\n");
}
```

**Case 3 and 8 are allowed to fall through to case 13**

**switch Example 3**

```c
switch(channel) {
  case 4 ... 7:  Apply this case to channel 4, 5, 6, and 7
    printf("VHF Station\n"); break;
  case 9 ... 12:
    printf("VHF Station\n"); break;
  case 3:  Case 3 and 8 are allowed to fall through to case 13
    printf("Weak Signal\n"); break;
  case 8:
  case 13:
    printf("Weak Signal\n"); break;
  case 14 ... 69:
    printf("UHF Station\n"); break;
  default:
    printf("No Signal Available\n");
}
```
Lab Exercise 6
Making Decisions: switch Statements

Exercise 06
Making Decisions (switch)

• Open the lab Project:

On the class website
Examples -> Lab6.zip

Open MPLAB®X and select Open Project Icon (Ctrl + Shift + O)
Open the Project listed above.

If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”
Exercise 06
Making Decisions (switch)

Solution: Step 1

```c
/*###########################################################################
# TASK: Write a switch statement to print the network's initials with the
#       channel (based on Chicago TV stations).
#       * If channel = 2, print "CBS 2" to the output window.
#       * If channel = 5, print "NBC 5" to the output window.
#       * If channel = 7, print "ABC 7" to the output window.
#       * For all other channels, print "--- #" to the output window,
#         where "#" is the channel number.
#       (HINT: Use printf(), and use the newline character '\n' at the end
#          of each string you print to the output window.)
# NOTE: The switch statement is in a loop that will execute 9 times. Each
#       pass through the loop, 'channel' will be incremented. The output
#       window should display a line of text for channels 2 to 10.
# # STEP 1: Open a switch statement on the variable 'channel'
############################################################################*/
 switch (channel) {
    
```
Exercise 06
Making Decisions (switch)

Solution: Steps 2 and 3

```c
/*###########################################################################
# STEP 2: Write case for channel = CBS (CBS is a constant defined to equal 2)
#----------------------------------------------------------------------*/
case CBS:
    //If channel = CBS (CBS = 2)
    {
        printf("CBS \%d\n", channel); //Display string "CBS 2" followed by newline
        break; //Prevent fall through to next case
    }

/*###########################################################################
# STEP 3: Write case for channel = NBC (NBC is a constant defined to equal 5)
#         This should look almost identical to step 2.
#----------------------------------------------------------------------*/
case NBC:
    //If channel = NBC (NBC = 5)
    {
        printf("NBC \%d\n", channel); //Display string "NBC 5" followed by newline
        break; //Prevent fall through to next case
    }
```

Solution: Steps 4 and 5

```c
/*###########################################################################
# STEP 4: Write case for channel = ABC (ABC is a constant defined to equal 7)
#         This should look almost identical to step 2.
#----------------------------------------------------------------------*/
case ABC:
    //If channel = ABC (ABC = 7)
    {
        printf("ABC \%d\n", channel); //Display string "ABC 7" followed by newline
        break; //Prevent fall through to next case
    }

/*###########################################################################
# STEP 5: Write default case. If channel is anything other than those
#         listed above, this is what should be done. For these cases, you
#         need to print the string "--- \#" where \"#\" is the channel number.
#         For example, if channel = 6, you should print "--- 6".
#----------------------------------------------------------------------*/
default:
    //For all other channels
    {
        printf("--- \%d\n", channel); //Display string "--- #" followed by newline
    }
```
Exercise 06
Conclusions

- `switch` provides a more elegant decision making structure than `if` for multiple conditions (if – else if – else if – else if…)
- The drawback is that the conditions may only be constants (match a variable's state to a particular value)

Loop Structures
**for Loop**

**Syntax**

```c
for (expression1; expression2; expression3)
    statement
```

- `expression1` initializes a loop count variable once at start of loop (e.g. `i = 0`)
- `expression2` is the test condition – the loop will continue while this is true (e.g. `i <= 10`)
- `expression3` is executed at the end of each iteration – usually to modify the loop count variable (e.g. `i++`)

---

**Flow Diagram**

```
START

Initialize loop variable
i = 0

expression1

Modify loop variable
i++

Test loop variable for exit condition
i < n

expression2?

TRUE

statement

FALSE

END
```
**for Loop**

### Example (Code Fragment)

```c
int i;
for (i = 0; i < 5; i++)
{
    printf("Loop iteration %d\n", i);
}
```

Expected Output:

```
Loop iteration 0
Loop iteration 1
Loop iteration 2
Loop iteration 3
Loop iteration 4
```

---

**Note**

- Any or all of the three expressions may be left blank (semi-colons must remain)
- If `expression_1` or `expression_3` are missing, their actions simply disappear
- If `expression_2` is missing, it is assumed to always be true

### Infinite Loops

A `for` loop without any expressions will execute indefinitely (can leave loop via `break` statement):  

```c
for ( ; ; )
{
    ...
}
```
**while Loop**

**Syntax**

```c
while (expression) statement
```

- If `expression` is true, `statement` will be executed and then `expression` will be re-evaluated to determine whether or not to execute `statement` again.

- It is possible that `statement` will never execute if `expression` is false when it is first evaluated.

---

**Flow Diagram**

Diagram showing the flow of a `while` loop:

- Start
- Evaluate `expression`
  - If `expression` is TRUE, execute `statement`
  - If `expression` is FALSE, end

---

Gabriel Hugh Elkaim – Spring 2013
**while Loop**

Example (Code Fragment)

```c
int i = 0;  // Loop counter initialized outside of loop
while (i < 5)  // Condition checked at start of loop iterations
{
    printf("Loop iteration %d\n", i++);  // Loop counter incremented manually inside loop
}
```

Expected Output:

- Loop iteration 0
- Loop iteration 1
- Loop iteration 2
- Loop iteration 3
- Loop iteration 4

**Note**

- The `expression` must always be there, unlike with a `for` loop.
- `while` is used more often than `for` when implementing an infinite loop, though it is only a matter of personal taste.
- Frequently used for main loop of program.

Infinite Loops

A `while` loop with `expression = 1` will execute indefinitely (can leave loop via `break` statement):

```c
while (1)  // infinite loop
{
    ...  // body of loop
}
```

- `while` Loop

- `expression` must always be there, unlike with a `for` loop.
- `while` is used more often than `for` when implementing an infinite loop, though it is only a matter of personal taste.
- Frequently used for main loop of program.