printf Semantics

- returns the number of characters that are output or a negative value if an output error occurs.
- Argument list expressions are converted according to the corresponding formatter in the control string.
- Control string formatters are indicated by preceding them with a % symbol.
- Argument list expressions and control string formatters are matched on a 1-1 basis.
- Text in the control string that is NOT part of a formatter is output verbatim.

Control String Formatters

<table>
<thead>
<tr>
<th>char</th>
<th>conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>decimal integer</td>
</tr>
<tr>
<td>%i</td>
<td>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</td>
</tr>
<tr>
<td>%c</td>
<td>character</td>
</tr>
<tr>
<td>%s</td>
<td>string or character array</td>
</tr>
<tr>
<td>%f</td>
<td>float</td>
</tr>
<tr>
<td>%e</td>
<td>double (scientific notation)</td>
</tr>
<tr>
<td>%g</td>
<td>float or double format</td>
</tr>
<tr>
<td>%%</td>
<td>outputs a % character</td>
</tr>
</tbody>
</table>

Learning by Example

- printf("The sum of %d, %d, and %d is %d\n", 65, 87, 33, 65+87+33);
  - The sum of 65, 87, and 33 is 185
- printf("Character code %c has ASCII code %d\n", 'A', 'A');
  - Character code A has ASCII code 65.
- printf("Error %s occurred at line %d \n", emsg, lno);
  - Error invalid variable occurred at line 27
  - Note: emsg and lno are variables!

C I/O

- Output to screen: printf

- Syntax
  - printf ( control string , argument list )
  - Function Prototype:
    - int printf( const char *format, ... );
  - Example:
    - char str[] = "world"; printf("hello %s", str);

Learning by Example

- printf("Octal form of %d is %o \n", 59, 59);
  - Octal form of 59 is 73
- printf("Hexadecimal form of %d is %x \n", 59, 59);
  - Hexadecimal form of 59 is 3B
- printf("Square root of 2 is %f \n", sqrt(2.0));
  - Square root of 2 is 1.414214

Learning by Example

- printf("The sum of %d, %d, and %d is %d\n", 65, 87, 33, 65+87+33);
  - The sum of 65, 87, and 33 is 185
- printf("Character code %c has ASCII code %d\n", 'A', 'A');
  - Character code A has ASCII code 65.
- printf("Error %s occurred at line %d \n", emsg, lno);
  - Error invalid variable occurred at line 27
  - Note: emsg and lno are variables!

Learning by Example

- printf("Octal form of %d is %o \n", 59, 59);
  - Octal form of 59 is 73
- printf("Hexadecimal form of %d is %x \n", 59, 59);
  - Hexadecimal form of 59 is 3B
- printf("Square root of 2 is %f \n", sqrt(2.0));
  - Square root of 2 is 1.414214

Learning by Example

- printf("The sum of %d, %d, and %d is %d\n", 65, 87, 33, 65+87+33);
  - The sum of 65, 87, and 33 is 185
- printf("Character code %c has ASCII code %d\n", 'A', 'A');
  - Character code A has ASCII code 65.
- printf("Error %s occurred at line %d \n", emsg, lno);
  - Error invalid variable occurred at line 27
  - Note: emsg and lno are variables!
Learning by Example

- `printf("Square root of 157 is %e \n", sqrt(157.0));`
  - Square root of 157 is 1.252996e+01
- `printf("You scored %d out of %d for \%d\%,.\n",17, 25, 68);`
  - You scored 17 out of 25 for 68%.

Flag Characters

- Optionally specified between the % and the formatter character
  - Field width
    - positive integer giving minimum number of output columns; effect depends upon output type
  - Precision
    - period followed by a nonnegative integer giving the minimum number of digits output for (d, i, o, u, x) or the digits to right of decimal point for (e, f) or maximum number of characters for (s)
  - Alignment
    - optionally precedes field width; minus sign specifies left-justification, no minus sign specifies right justification

Miscellaneous Flags

- plus sign optionally preceding field width specifies that a ‘+’ character is prefixed to nonnegative numbers
- pound sign optionally preceding field width specifies that a zero be prefixed to octals and a 0X be prefixed to hexadecimals
- a zero optionally preceding field width specifies that zeroes are to be used for padding instead of spaces

Longs and Shorts

- h character optionally preceding field width specifies that a conversion to short int occurs if required prior to output
- l or L character optionally preceding field width specifies that a conversion to long int or long double respectively occurs if required prior to output

Other Characters

<table>
<thead>
<tr>
<th>code</th>
<th>control character</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln</td>
<td>new line</td>
</tr>
<tr>
<td>t</td>
<td>tab</td>
</tr>
<tr>
<td>lr</td>
<td>carriage return</td>
</tr>
<tr>
<td>f</td>
<td>form feed</td>
</tr>
<tr>
<td>lv</td>
<td>vertical tab</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
<tr>
<td>a</td>
<td>alert (bell)</td>
</tr>
<tr>
<td>b</td>
<td>backspace</td>
</tr>
</tbody>
</table>

The `printf()` & `scanf()` functions are located in the standard I/O library: 

`#include <stdio.h>`

Input: Using `scanf`

- Syntax
  - `scanf( control string , argument list )`
  - Function Prototype: `int scanf( const char *format, ... );`
- Semantics
  - Returns number of variables assigned values or EOF if an error.
  - Argument list expressions are pointer expressions converted according to the corresponding formatter in the control string.
  - Standard input file stdin (keyboard) supplies the input stream.
Primary differences: printf() / scanf() formatter chars
- `%c` specifier can match white space characters in the input.
- White space characters in the control string (`t \n`) which can match optional white space in the input.
- Nonwhite space characters (not `%`), which must match the next input character in the stream.
- `\n` causes no input to occur, scanf stores the number of characters read so far by the current scanf() in the corresponding integer pointer argument.

More Differences
- `%%` instructs scanf to skip a `%` sign in the input.
- The asterisk character (suppression character `*`) preceding a formatter character instructs scanf() to discard the corresponding input data without storing it in a variable.
  - e.g., `%c` would cause scanf() to discard the next input character.

More differences
- Field width specifiers: scanf() inputs multiple characters:
  - e.g., `%s` would cause scanf() to skip white space, read in nonwhite space characters stopping at the next white space character.
  - e.g., `%6s` causes scanf() to skip white space, read the next 6 nonwhite input characters (stopping at the next white space or end of file).
  - e.g., `%6c` causes scanf() to read the next 6 input characters, including whitespace, (or stopping at end of file).

Learning by Example
- scanf(`"%d%i%i%i%o%x"`, &a1, &a2, &a3, &a4, &a5, &a6);
  - /* Input echoed as decimal ints */
  - printf(`"%d %d %d %d %d %d\n"`, a1, a2, a3, a4, a5, a6);

Input Data

...into that good night

Output

"... into that good night"

Learning by Example
- scanf(`"%c%c%s"`, &c1, &c2, s);
  - /* Input echo */
  - printf(`"%c%c %s\n"`, c1, c1, s);

Final Thoughts:
- printf() & scanf() have options other than those covered herein.
- Knowledge of printf() & scanf() is required to understand legacy C code, but they should be avoided when possible.
- fprintf and fscanf are the file versions of the same calls: see website for slides.
Multiple Arguments

- If you’ve noticed, we obviously can input an undefined number of variables to the printf and scanf functions…HOW?
- `#include <stdarg.h>` gives some help!

Variable # Function Parameters

- Specifying functions that accept an unknown number of arguments.
- Ellipsis “…” in a function prototype indicates the function accepts a arbitrary number of parameters.

```
int total ( int , ... );
```

- At least one named parameter must be specified.
- The ellipsis must be placed at the end of the parameter list.

STDARG.H

- The predefined types that support variable length parameter lists is defined in “stdarg.h”. Contains one type definition and three macro functions.

```
va_list va_start( va_list, int);
```

- The first argument must be of type `va_list` to hold the variable number of parameters.
- The second argument should be of type `int` and must be the last parameter in the variable-length function heading, (i.e., immediately preceding the ellipsis. When called it holds the count of the number of arguments.)

```
type va_arg(va_list, type);
```

- The first argument must be of type `va_list` holding the variable number of parameters list previously returned by a call to `va_start()`.
- The second argument must be a standard C/C++ predefined language type.
- The call returns the next argument in the list, converted to the passed type.

stdarg.h Declarations:

- `va_list` Predefined type for storing the variable length argument list.
- `va_start` Macro function called to obtain the variable length argument list.
- `va_arg` Macro function that returns a value in the argument list.
- `va_end` Macro function called to perform necessary (memory) cleanup before function exits.
void va_end (va_list);

- The first argument must be of type va_list holding the variable number of parameters list previously returned by a call to va_start().

```c
#include <stdarg.h>

double total( int c, . . . ) {
  va_list v;
  double sum=0.0;
  va_start(v, c);
  for (int i=0; i < c; i++)
    sum += va_arg(v, double);
  va_end(v);
  return( sum );
}
```

```c
double pi=3.14159, e=2.71828, myTotal;
int args=2;
myTotal = total(args, pi, e);
myTotal = total(3, -1.0, -2.0, -3.0);
```