Using Program Memory for Storage - work in progress

Thanks to Dean Cammera for his excellent tutorial on using program memory. Most of this work was derrived from his tutorial.

Sometimes our programs have large tables or many strings to store. If we declare a string like this:

```
char error_code1[7] = {"error1"};
```

Or, an inline string:

```
uart-puts("This is my newest shaded string.");
```

gcc will by default, put the string into RAM as initalized data storage using the startup code. This makes sense for workstation programs as string functions operate on pointers to strings located in RAM.

Using Program Memory for Storage

However, on most microcontrollers, RAM is scarce and program (flash) memory is plentiful. It is often helpful to store constant data in program memory and perserve RAM. AVR Libc provides a set of functions and macros that allow us to do that. To enable this functionality we declare:

```
#include <avr/pgmspace.h>
```

To have gcc store our string into program memory, we can say:

```
const char error_code1[7] PROGMEM = {"error1"};
```

The PROGMEM attribute modifier forces the string to be stored into program memory.

Using Program Memory for Storage

The library progmem.h also includes a macro "PSTR" that allows us to create an inline string stored in program memory.

```
uart_puts(PSTR("This is my string"));
```

The string "This is my string" will be stored in program memory.

Note:
The PSTR macro can only be used within functions.
The compiler treats all PSTR declared strings as separate entities.

Using Program Memory for Storage

If we create strings in program memory, our usual functions that expect pointers to RAM memory will not work. We need a way to make our functions understand that the pointers are pointing to program memory. For example, suppose you have the following UART function:

```
void uart_puts(char *str){
  while(*str != '\0'){
    uart_putc(*str++);
  }
}
```

The function `uart_puts` will not understand that the pointer passed to it is a pointer to program memory.

Using Program Memory for Storage

progmem.h provides a function `pgm_read_byte()` that returns a byte located at the pointer passed to it. Therefore, we can rewrite our function as follows:

```
void uart_puts(char *str){
  while(pgm_read_byte(str) != '\0'){
    uart_putc(pgm_read_byte(str++));
  }
}
```

Of course if you needed a function to read from both program memory and from ram you would need two different named functions.

Using Program Memory for Storage

As mentioned before, the progmem.h library allows us to also store data arrays.

```
static unsigned int LCD_font_table[] PROGMEM{
  0xABCD,
  0x1234,
  0xDEAD,
  0x9876
}
```

Since this data is 16 bits long, we can retrieve the data using the progmem.h function `read_pgm_word()`.
To access the first element of the array, we could say:
`pgm_read_word{&LCD_font_table[0])`