要想将数组中保存的字符串存到flash，若该数组为全局变量，用法是：

1. const char PROGMEM str[]="ABCDE";

若该数组为局部变量，需要配合static关键词使用，用法是：

1. static const char str[] PROGMEM = "ABCDE";

也可以使用PSTR()将字符串保存到flash，但是应注意点：

①，只能使用指针指向PSTR()，不能使用数组。例如下面的写法是正确的：

const char \*str=PSTR("FGHIJ");

但是不能这样写：

const char str[]=PSTR("FGHIJ");

②，不能加static修饰。

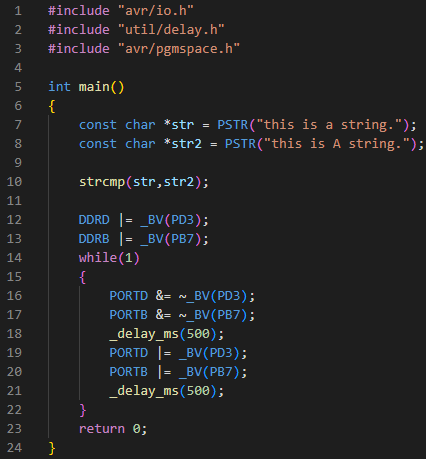
③，只能写在函数中，不能写在函数外。

当向一个函数传递指向flash的指针时，编译器会认为该指针指向ram，从而从ram中寻找数据，使程序出错。为解决这一问题，需要用专门的函数来处理指向flash的指针。

标准C语言提供了一些字符串操作函数，例如：

1. void \*memcpy(void \*, const void \*, size\_t);
2. char \*strcat(char \*, const char \*);
3. int strcmp(const char \*, const char \*)；
4. char \*strcpy(char \*, const char \*);

这些函数的形参是指针，但是，若在AVR C语言中直接使用上述标准函数，例如：



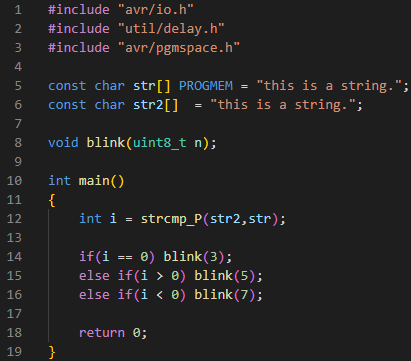
这样使用strcmp()函数并不会比较str与str2字符串，因为str与str2是指向FLASH的地址，strcmp()在接收到这两个地址后，并不会认为接收的地址是FLASH的地址，而是认为接收的地址是SRAM地址，进而从SRAM中取对应的数据，这不是我们期望的。为此，avr-libc提供了与标准C语言对应的函数：

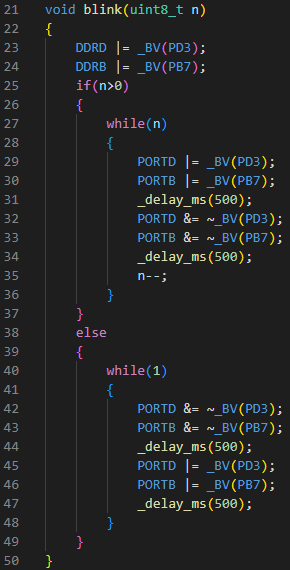
1. void \*memcpy\_P(void \*, const void \*, size\_t);
2. char \*strcat\_P(char \*, const char \*);
3. int strcmp\_P(const char \*, const char \*)；
4. char \*strcpy\_P(char \*, const char \*);

还以strcmp\_P()为例，查看其帮助文档：



该函数的第一个参数必须是指向SRAM的指针，第二个参数必须是指向FLASH的指针，只有这样，strcmp\_P()的结果才正确。例如：





上面的程序中，str[]定义在FLASH中，而str2[]定义在SRAM中。程序运行结果是LED灯闪烁3次，正确。由此可见，使用avr-libc的函数，必须仔细阅读其函数说明。