LANGUAGE
FUNCTIONS
VARIABLES
STRUCTURE
LIBRARIES
IOT CLOUD API
GLOSSARY

Last Revision: 2022/07/07

Last Build: 2023/05/20

EDIT THIS PAGE

Reference > Language > Variables > Utilities > Progmem

PROGMEM

[Utilities]

Description

Store data in flash (program) memory instead of SRAM. There's a description of the various types of memory available on an Arduino board.

The `PROGMEM` keyword is a variable modifier, it should be used only with the datatypes defined in pgmspace.h. the compiler "put this information into flash memory", instead of into SRAM, where it would normally go.

PROGMEM is part of the pgmspace.h library. It is included automatically in modern versions of the IDE. Howe you are using an IDE version below 1.0 (2011), you'll first need to include the library at the top of your sketch, this:

`#include <avr/pgmspace.h>` While `PROGMEM` could be used on a single variable, it is really only worth the fuss if y have a larger block of data that needs to be stored, which is usually easiest in an array, (or another C++ data structure beyond our present discussion).

Using `PROGMEM` is also a two-step procedure. After getting the data into Flash memory, it requires special meth (functions), also defined in the pgmspace.h library, to read the data from program memory back into SRAM, s can do something useful with it.

Syntax

```
const dataType variableName[] PROGMEM = {data0, data1, data3…};
```

Note that because PROGMEM is a variable modifier, there is no hard and fast rule about where it should go, s Arduino compiler accepts all of the definitions below, which are also synonymous. However, experiments hav indicated that, in various versions of Arduino (having to do with GCC version), PROGMEM may work in one lo and not in another. The "string table" example below has been tested to work with Arduino 13. Earlier versio the IDE may work better if PROGMEM is included after the variable name.

```
const dataType variableName[] PROGMEM = {}; // use this form
const PROGMEM dataType variableName[] = {}; // or this one
const dataType PROGMEM variableName[] = {}; // not this one
```

Parameters

`dataType`: Allowed data types: any variable type.
`variableName`: the name for your array of data.

Example Code

The following code fragments illustrate how to read and write unsigned chars (bytes) and Help PROGMEM.

```
unsigned int displayInt;
char myChar;


void setup() {
  Serial.begin(9600);
  while (!Serial);  // wait for serial port to connect. Needed for native USB

  // put your setup code here, to run once:
  // read back a 2-byte int
  for (byte k = 0; k < 5; k++) {
    displayInt = pgm_read_word_near(charSet + k);
    Serial.println(displayInt);
  }
  Serial.println();

  // read back a char
  for (byte k = 0; k < strlen_P(signMessage); k++) {
    myChar = pgm_read_byte_near(signMessage + k);
    Serial.print(myChar);
  }

  Serial.println();
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Arrays of strings

It is often convenient when working with large amounts of text, such as a project with an LCD, to setup an arr strings. Because strings themselves are arrays, this is actually an example of a two-dimensional array.

These tend to be large structures so putting them into program memory is often desirable. The code below illustrates the idea.

```
/*
  PROGMEM string demo
  How to store a table of strings in program memory (flash),
  and retrieve them.

  Information summarized from:
  http://www.nongnu.org/avr-libc/user-manual/pgmspace.html

  Setting up a table (array) of strings in program memory is slightly complicated, but
  here is a good template to follow.

  Setting up the strings is a two-step process. First, define the strings.
*/

#include <avr/pgmspace.h>
const char string_0[] PROGMEM = "String 0"; // "String 0" etc are strings to store - change to suit.
const char string_1[] PROGMEM = "String 1";
const char string_2[] PROGMEM = "String 2";
const char string_3[] PROGMEM = "String 3";
const char string_4[] PROGMEM = "String 4";
const char string_5[] PROGMEM = "String 5";


// Then set up a table to refer to your strings.

const char *const string_table[] PROGMEM = {string_0, string_1, string_2, string_3, string_4, string_5};

char buffer[30];  // make sure this is large enough for the largest string it must hold

void setup() {
  Serial.begin(9600);
  while (!Serial);  // wait for serial port to connect. Needed for native USB
  Serial.println("OK");
}


void loop() {
  /* Using the string table in program memory requires the use of special functions to ret[illegible]ata.
     The strcpy_P function copies a string from program space to a string in RAM ("buffer").
```

Help

```
    Serial.println(buffer);
    delay(500);
  }
}
```

Notes and Warnings

Please note that variables must be either globally defined, OR defined with the static keyword, in order to wo PROGMEM.

The following code will NOT work when inside a function:

```
const char long_str[] PROGMEM = "Hi, I would like to tell you a bit about myself.\n";
```

The following code WILL work, even if locally defined within a function:

```
const static char long_str[] PROGMEM = "Hi, I would like to tell you a bit about myself.\n"
```

The `F()` macro

When an instruction like :

```
Serial.print("Write something on  the Serial Monitor");
```

is used, the string to be printed is normally saved in RAM. If your sketch prints a lot of stuff on the Serial Mon you can easily fill the RAM. If you have free FLASH memory space, you can easily indicate that the string must saved in FLASH using the syntax:

```
Serial.print(F("Write something on the Serial Monitor that is stored in FLASH"));
```

See also

EXAMPLE Types of memory available on an Arduino board

DEFINITION array

DEFINITION string