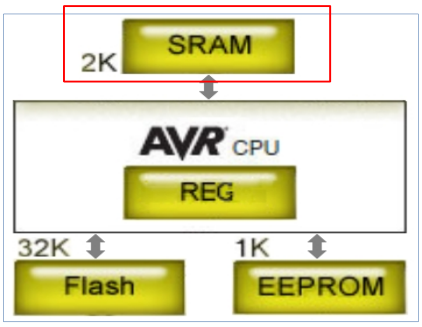
ATMEGA328/Arduino Uno – MEMORY – VARIABLE – PART 1

https://github.com/teaksoon/lmaewapm

WORKING MEMORY

The “WORKING MEMORY” is for us to use in our PROGRAM as temporary working storage

This MEMORY can be visualized as a long sequence of individual BITs



To use the **WORKING MEMORY (SRAM)** in our PROGRAM, we will need “Reserve in BLOCKS of multiple BITS”. Each “BLOCK” of “Reserved MEMORY” is commonly known as **“VARIABLE”**

The C-Language Keywords and Symbols

Keywords

MEMORY	CONTROL
01.void	21.return
02.char	22.if
03.int	23.else
04.short	24.switch
05.long	25.case
06.float	26.default
07.double	27.while
08.signed	28.do
09.unsigned	29.for
10.struct	30.break
11.union	31.continue
12.enum	32.goto
13.const	
14.volatile	
15.auto	
16.extern	
17.static	
18.register	
19.typedef	
20.sizeof	

Symbols

CONTROL	LOGIC	MATH	BIT OP
#	==	*	|
< >	!=	%	&
//	<	/	^
/* */	>	+	~
( )	<=	-	<<
{ }	>=		>>
;	&&		
,	||		
“			
‘	!		
=			
[ ]			
:			
?			
.			
\			
MEMORY			
&			
*			

Out of the Total 32 C-Language Keywords
20 Keywords are used to handle the WORKING MEMORY

- The **first 18 Keywords**, are used exclusively for creating “VARIABLES” with its various properties and features

19.typedef – utility for us to create new “datatype” from the existing “datatype”
20.sizeof – utility for us to find out the number of BYTE used in any “variable”

That leaves us just **12 Keywords for all the other things.** This is how important the WORKING MEMORY is to C-Language

Lets look at all the **20 C-Language Keywords** that we can use **to PROGRAM “VARIABLE” with its various properties and features**

BASIC DATATYPE

These are the BASIC DATATYPE used in C-Language

02.char (8-BIT)
03.int (16-BIT, can be different for non 8-BIT micro-controllers)
04.short (16-BIT, normally do not use this, it is slower than int)
05.long (32-BIT)
06.float (32-BIT)
07.double (32-BIT, for ATMEGA328, it is same as float, we use float)

08.signed (The stored number can have **Positive and Negative Numbers**)
09.unsigned (The stored number can have **Positive Numbers Only**)

EXTENDED DATATYPE

These are extensions to the BASIC DATATYPE

10.struct (combinations of multiple BASIC DATATYPE/EXTENDED DATATYPE)
11.union (multiple datatype “over-lapping” each other on the same MEMORY)
12.enum (16-BIT number in sequence, each referenced with a “name”)

PROPERTIES

Also known as type qualifiers

13.const (Data cannot be changed, Initial Data can be assigned)
17.volatile (Prevent the Compiler from Automatic Optimizing of MEMORY)

FEATURES

These are Storage Classes of a variable

15.auto (DATATYPE follow the initial Data assigned)
16.extern (DATATYPE declaration, when code is stored in different file)
14.static (Data can be changed, Initial Data can only be assigned just ONCE)
18.register (Use CPU MEMORY instead of WORKING MEMORY) *compiler decides

DATATYPES UTILITIES

19.typedef (make new DATATYPE from the existing DATATYPE)
20.sizeof (find out the number of BYTE in any DATATYPE) : 1 BYTE = 8 BITS

VOID

01.void (Empty datatype)
- This is most commonly used as function return datatype when function returns nothing
- This is less commonly used as parameter(optional) when functions that does not have any parameters and for memory pointers to unknown datatype

There the 20 Keywords from the C-Language that we can use to manipulate the WORKING MEMORY
We will start with the commonly used ones

BASIC DATATYPE: Create Variable
Part1:datatype
Part2:name

Part1: datatype
– datatype, followed by space

Part2:name
– name followed by a semi-colon ;

```
datatype name;
```

BASIC DATATYPE: Create Variable with initial value
Part1:datatype
Part2:name
Part3:value

Part1:datatype
– datatype, followed by space

Part2:name
– name followed by an equal sign =

Part3:value followed by a semi colon ;

```
datatype name = value;
```

Example PROGRAM:

Arduino IDE|Save PROGRAM as: **c_variable_basic**
Enter codes below and upload. Use the Serial Monitor to see results

```
int counter = 0;
int temp;
float frac_num = 1.5;

void setup(){
  Serial.begin(9600);
  Serial.print("\nSerial Monitor at 9600 baud...\n");

  Serial.print("\nData stored in variable counter = ");
  Serial.print(counter);
  temp = counter+1;
  Serial.print("\nData stored in variable temp = ");
  Serial.print(temp);
  Serial.print("\nData stored in variable frac_num = ");
  Serial.print(frac_num,2);
}
void loop(){}
```

NOTE: BASIC DATATYPES, “char,int,long” stores whole numbers, while “float” stores numbers with fractions
02.char
03.int (**04.short** is the same, normally we use “int”)
05.long
06.float (**07.double** is the same, normally we use “float”)

BASIC DATATYPE: Create Variable with "unsigned" Keyword
Part1:"unsigned" Keyword
Part2:datatype
Part3:name

Part1:"**unsigned**" Keyword followed by space

Part2:datatype
– datatype, followed by space

Part3:name
– name followed by a semi-colon ;

```
unsigned datatype name;
```

Example PROGRAM:

Arduino IDE|Save PROGRAM as: **c_variable_unsigned**
Enter codes below and upload. Use the Serial Monitor to see results

```
unsigned int counter;

void setup(){
  Serial.begin(9600);
  Serial.print("\nSerial Monitor at 9600 baud...\n");

  counter =  1; // Positve Number
  Serial.print("\n1 stored in unsigned variable counter = ");
  Serial.print(counter);

  counter = -1; // Negative Number ( this is not allowed )
  // -1 will be converted into positive number
  // "int" datatype use twos complement for negative numbers
  // therefore, -1 will be converted to 65535
  Serial.print("\n-1 stored in unsigned variable counter = ");
  Serial.print(counter);
}
void loop(){}
```

NOTE: if the "unsigned" Keyword is not specified, the variable is "signed" number by default

– char,int,long – stores negative whole numbers in "twos complement" format
– float – stores numbers with fraction in "IEEE 754" format

08.signed (The stored number can have **Positive and Negative Numbers**)
09.unsigned (The stored number can have **Positive Numbers Only**)

BASIC DATATYPE: Create Variable with “const” Keyword
Part1:”const” Keyword
Part2:datatype
Part3:name
Part4:value

Part1:”const” Keyword followed by space

Part2:datatype
– datatype, followed by space

Part3:name
– name followed by an equal sign =

Part4:value
– value followed by a semi-colon ;

```
const datatype name = value;
```

Example PROGRAM:

Arduino IDE|Save PROGRAM as: **c_variable_const**
Enter codes below and upload. Use the Serial Monitor to see results

```
const int led_pin = 9;

void setup(){
  Serial.begin(9600);
  Serial.print("\nSerial Monitor at 9600 baud...\n");

  Serial.print("\nData stored in const variable led_pin = ");
  Serial.print(led_pin);

  // the following code will compile with error, un-comment to test
  // led_pin = 10;
}

void loop(){}
```

NOTE: A const variable must be assigned an initial value. After that, the assigned initial value cannot be changed

13.const

BASIC DATATYPE: Create Variable with “static” Keyword
Part1:”static” Keyword
Part2:datatype
Part3:name
Part4:value

Part1:”static” Keyword followed by space

Part2:datatype
- datatype, followed by space

Part3:name
- name followed by an equal sign =

Part4:value
- value followed by a semi-colon ;

```
static datatype name = value;
```

Example PROGRAM:

Arduino IDE|Save PROGRAM as: **c_variable_static**
Enter codes below and upload. Use the Serial Monitor to see results

```
void setup(){
  Serial.begin(9600);
  Serial.print("\nSerial Monitor at 9600 baud...\n");
}

void loop(){
int temp;
  temp = static_var();
  Serial.print("\n\nData from static_var() function = ");
  Serial.print(temp);
  temp = non_static_var();
  Serial.print("\nData from non_static_var() function = ");
  Serial.print(temp);
  delay(2000);
}

int static_var(){
static int counter = 0;
  counter = counter+1;
  return counter;
}

int non_static_var(){
int counter = 0;
  counter = counter+1;
  return counter;
}
```

NOTE: Need to specify an initial value. When used in a function, this value will not reset with initial value when the function is called again, it will retain whatever value from the previous function call

14.static