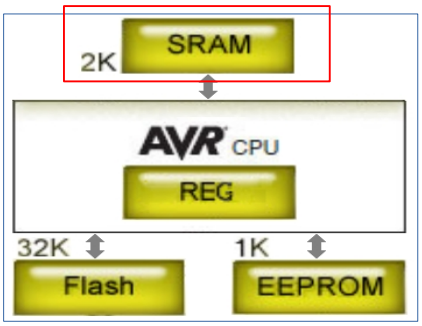
ATMEGA328/Arduino Uno – MEMORY – VARIABLE – PART 2
https://github.com/teaksoon/lmaewapm

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 just handle the WORKING MEMORY

– The **first 18 Keywords,** are used exclusively for creating/declaring VARIABLES with its various properties and features

19.typedef – utility for us to create a 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 the **20 C-Language Keywords** that we use to manipulate the WORKING MEMORY

(PART 2)
This part is mostly about the SPECIAL DATATYPE
(We may not even need to use them at all, we keep this just in case)

BASIC DATATYPE
these are the BASIC DATATYPE used in C-Language

02.char (8-BIT)
03.int (16-BIT for ATMEGA328P) - can be different for others
04.short (16-BIT)
05.long (32-BIT)
06.float (32-BIT)
07.double (32-BIT for ATMEGA328P) - can be different for others

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

SPECIAL DATATYPE
these are extensions to the BASIC DATATYPE

10.struct (Multiple members of various DATATYPE)
11.union (Multiple members of various DATATYPE sharing the same the MEMORY)
12.enum (Multiple members of 16-BIT auto-increment number)

PROPERTIES
also known as type qualifiers

13.const (Variable with Initial Data assigned that cannot be changed)
14.volatile (Prevent the Compiler from automatic Optimizing of MEMORY)

FEATURES
these are Storage Classes of a Variable

15.auto (DATATYPE from the Initial Data assigned)
16.extern (Variable declaration is stored in a different file)
17.static (Variable with Initial Data assigned just once, retains value)
18.register (Use CPU MEMORY instead of WORKING MEMORY) *compiler decides

DATATYPE UTILITIES

19.typedef (Create a new DATATYPE from the existing DATATYPE)
20.sizeof (Find out the number of BYTE in any Variable) : 1 BYTE = 8 BITS

VOID

01.void (Empty DATATYPE)
- Most commonly used as function return DATATYPE when a function returns nothing
- Less commonly used as parameter(optional) when functions that does not have any parameters
- Also used as memory pointers Variable to unknown DATATYPE

DATATYPE UTILITY: "typedef" to Create NEW DATATYPE from existing DATATYPE
Part1:"typedef" Keyword
Part2:datatype
Part3:new_datatype

Part1:"typedef" Keyword,followed by space

Part2:datatype, existing datatype followed by space

Part3:new_datatype, followed by semi-colon ;

```
typedef datatype new_datatype;
```

Example PROGRAM:

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

```
typedef unsigned char on_off;

void setup(){
  on_off led_state = 0; // using our new DATATYPE "on_off"

  Serial.begin(9600);Serial.print("\nSerial Monitor(9600)...\n");

  led_state = 1;
  Serial.print("\nData stored in Variable led_state = ");
  Serial.print(led_state);

  Serial.print("\nTotal BITS in Variable led_state = ");
  Serial.print(sizeof(led_state)*8);
}
void loop(){}
```

NOTE: we have created a new DATATYPE called "on_off" by using the "typedef" Keyword.

In this example PROGRAM:
The Variable "led_state" is Declared using the new DATATYPE "on_off". We can use this Variable just like any other Variables. In this same PROGRAM, we also use the **"sizeof" Keyword** to see the number of BITS used in our "led_state" Variable

C-Language Keywords
19.typedef (Create a new DATATYPE from the existing DATATYPE)
20.sizeof (Find out the number of BYTE in any Variable) : 1 BYTE = 8 BITS

"struct" has two part process. Part 1(similar to "typedef")
Part1:"struct" Keyword
Part2:struct_name
Part3:body

Part1:"struct" Keyword, followed by space
Part2:struct_name

Part3:body
- within a curly bracket { } pair
- One or More "members"
- Each "members" is a Variable Declaration
- semi-colon ; after the closing curly bracket }

```
struct struct_name
{
  datatype name;
  // other members;
};
```

"struct" has two part process. Part 2(struct_name is used like a DATATYPE)
Part1:"struct" Keyword
Part2:struct_name
Part3:name

Part1:"struct" Keyword,followed by space
Part2:struct_name, followed by space
Part3:name, followed by semi-colon ;

```
struct struct_name name;
```

Example PROGRAM:

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

```
struct my_position {
  int row;
  int col;
};
void setup(){
  Serial.begin(9600); Serial.print("\nSerial Monitor(9600)...\n");

  struct my_position pos;
  // my_position pos; // in newer C-Language, we can also code this way
  pos.row = 5;
  pos.col = 10;
  Serial.print("\nData stored in variable pos.row = ");
  Serial.print(pos.row);
  Serial.print("\nData stored in variable pos.col = ");
  Serial.print(pos.col);
  Serial.print("\nTotal BITS in variable pos = ");
  Serial.print(sizeof(pos)*8);
}
void loop(){}
```

NOTE: The "struct" have multiple-members in them where each member is a Variable Declaration. We access the members by using a dot . Symbol. In this example: "pos" is our Variable name from my_position "struct". We code "pos.row" and "pos.col" to access to the members in this "struct"

C-Language Keywords
10.struct (Multiple members of various DATATYPE)

"union" has two part process. Part 1(similar to "typedef")
Part1:"union" Keyword
Part2:union_name
Part3:body

Part1:"union" Keyword, followed by space
Part2:union_name

Part3:body
- within a curly bracket { } pair
- Two or More "members"
- Each "members" is a Variable Declaration
- semi-colon ; after the closing curly bracket }

```
union union_name
{
  datatype name;
  // other members;
};
```

"union" has two part process. Part 2(union_name is used like a DATATYPE)
Part1:"union" Keyword
Part2:union_name
Part3:name

Part1:"union" Keyword,followed by space
Part2:union_name, followed by space
Part3:name, followed by semi-colon ;

```
union union_name name;
```

Example PROGRAM:

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

```
union memory_16bit {
  int whole_16bit;
  struct whole_split {
    char part_b_8bit;
    char part_a_8bit;
  } split;
};
void setup(){
  Serial.begin(9600);Serial.print("\nSerial Monitor(9600)...\n");
  union memory_16bit mdata;

  mdata.whole_16bit = 259;
  Serial.print("\nData stored in whole_16bit = ");
  Serial.print(mdata.whole_16bit);Serial.print("\n");
  for (int i=15; i>=0; i--) {
    Serial.print((mdata.whole_16bit >> i) & 1);Serial.print(" ");
  }
  Serial.print("\n\nData stored in part_b_8bit = \n");
  for (int i=8; i>=0; i--) {
    Serial.print((mdata.split.part_b_8bit >> i) & 1);Serial.print(" ");
  }
}
void loop(){}
```

NOTE: The "union" have multiple-members in them where each member is a Variable Declaration. The two members in this example shares the same MEMORY

C-Language Keywords
11.union (Multiple members of various DATATYPE sharing the same the MEMORY)

"enum" has two part process. Part 1(similar to "typedef")
Part1:"enum" Keyword
Part2:enum_name
Part3:body

Part1:"enum" Keyword, followed by space
Part2:enum_name

Part3:body
- within a curly bracket { } pair
- One or More "members"
- Each "members" is a name seperated by comma , Last member does not need comma ,
- semi-colon ; after the closing curly bracket }

```
enum enum_name
{
  member_one_name,
  // other members
};
```

"enum" has two part process. Part 2(enum_name is used like a DATATYPE)
Part1:"enum" Keyword
Part2:enum_name
Part3:name

Part1:"enum" Keyword,followed by space
Part2:enum_name, followed by space
Part3:name, followed by semi-colon ;

```
enum enum_name name;
```

Example PROGRAM:

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

```
enum day_of_week {
  Mon=1,Tue,Wed,Thu,Fri,Sat,Sun
};

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

  enum day_of_week dow;

  dow = Tue; // here we specify the enum member name
  Serial.print("\nValue Stored in dow = ");Serial.print(dow);

  dow = Sun; // Sun is a value auto-increment from previous member
  Serial.print("\nValue Stored in dow = ");Serial.print(dow);

}
void loop(){}
```

NOTE: The "enum" have multiple-members in them where each member is a "name". Each member holds a 16-BIT number, auto-incremented by one from previous member.

In this example we have a Variable "dow" from "day_of_week" enum. The "dow" Variable accept assignment using the enum member name, making our PROGRAM source codes more readable

C-Language Keywords
12.enum (Multiple members of 16-BIT auto-increment number)