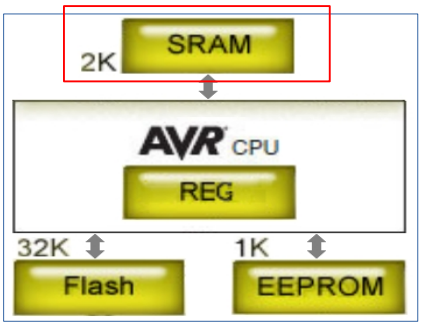
ATMEGA328/Arduino Uno – MEMORY – VARIABLE – PART 3
https://github.com/teaksoon/lmaewapm

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

ATMEGA328/Arduino Uno – MEMORY – VARIABLE – PART 3
https://github.com/teaksoon/lmaewapm

Lets look at the **20 C-Language Keywords** that we use to manipulate the WORKING MEMORY

(PART 3)
The remaining 4 C-Language Keywords for MEMORY
(for simple PROGRAM we probably will not use them)

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

BASIC DATATYPE: Create/Declare Variable with "auto" Keyword
Part1:"auto" Keyword
Part2:name
Part3:value

Part1:"auto" Keyword followed by space
Part2:name, name followed by an equal sign =
Part3:value, followed by a semi-colon ;

```
auto name = value;
```

Example PROGRAM:

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

```
auto counter = 0;
auto temp = 'A';
auto millis_timer = millis();

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

  Serial.print("\nData stored in variable counter = ");
  Serial.print(counter);
  Serial.print("\nNumber of BITS used by counter = ");
  Serial.print(sizeof(counter)*8);

  Serial.print("\n\nData stored in variable temp = ");
  Serial.print(temp);
  Serial.print("\nNumber of BITS used by temp = ");
  Serial.print(sizeof(temp)*8);

  Serial.print("\n\nData stored in variable millis_timer = ");
  Serial.print(millis_timer);
  Serial.print("\nNumber of BITS used by millis_timer = ");
  Serial.print(sizeof(millis_timer)*8);
}
void loop(){}
```

NOTE: When we use "auto" we do not need to specify the DATATYPE. However, an initial value must be provided during Variable declaration. Based on the initial value assigned, a suitable DATATYPE is chosen.

In this example PROGRAM:
- temp variable is assigned with 'A', which is a char DATATYPE. Therefore it is 1-BYTE
- millis_timer variable is assigned with value return from millis() function, which is "long" DATATYPE. Therefore it is 4-BYTE

"auto" Keyword is seldom used because it can cause confusion in our PROGRAM

For example: auto counter = 0, in this case auto uses 2-BYTE which is an "int". "long","char" and "short" are also valid DATATYPE

C-Language Keyword
15.auto (DATATYPE from the Initial Data assigned)

ATMEGA328/Arduino Uno – MEMORY – VARIABLE – PART 3
https://github.com/teaksoon/lmaewapm

BASIC DATATYPE: Create/Declare Variable with "extern" Keyword
Part1:"extern" Keyword
Part2:name
Part3:value

Part1:"extern" Keyword followed by space
Part2:datatype, followed by space
Part3:name, followed by a semi-colon ;

```
extern datatype name;
```

Or with an initial value

```
extern datatype name = 0;
```

Example PROGRAM:

Arduino IDE|Save PROGRAM as: **c_variable_extern**
Enter codes below

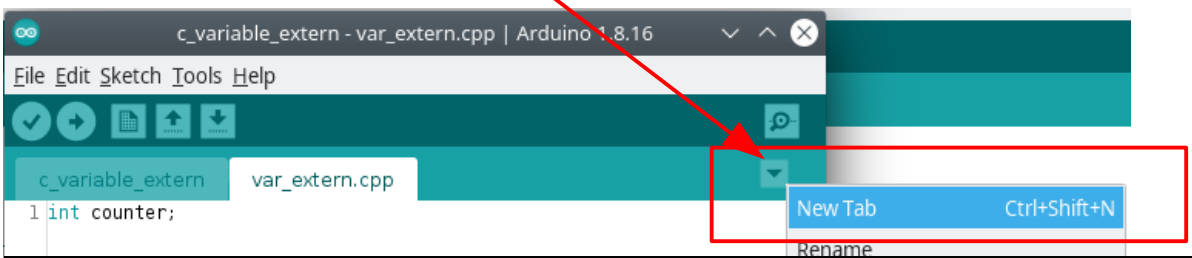
```
extern int counter;

void setup(){
  counter = 0;
  Serial.begin(9600);Serial.print("\nSerial Monitor(9600)...\n");

  Serial.print("\nData stored in variable counter = ");
  Serial.print(counter);
  Serial.print("\nNumber of BITS used by counter = ");
  Serial.print(sizeof(counter)*8);

}
void loop(){}
```

Arduino IDE|Create a new tab, Name it: **var_extern.cpp**
Enter codes below




```
int counter;
```

NOTE: This is exactly same like the regular Variable Declaration. Usage of the Variable from this Declaration is also the same.

It is just telling the Compiler Software that this Variable is declared somewhere else, in another file. In this example, the counter variable is declared in another file name "var_extern.cpp". Try comment off the extern and see what happens

C-Language Keyword
16.extern (Variable declaration is stored in a different file)

BASIC DATATYPE: Create/Declare Variable with "volatile" Keyword
Part1:"volatile" Keyword
Part2:name
Part3:value

Part1:"volatile" Keyword followed by space
Part2:datatype, followed by space
Part3:name, followed by a semi-colon ;

```
volatile datatype name;
```

Or with an initial value

```
volatile datatype name = 0;
```

Example PROGRAM:

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

```
volatile const int data = 5;  // try change these two to see impact
// const int data = 5;

void setup() {
int *ptr = (int*) &data;

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

  Serial.print("\nInitial value of data = "); Serial.print(data);

  *ptr = 20; // Without "volatile" Compiler optimize this (getting rid of
this code)

  Serial.print("\nNew value of data = ");Serial.print(data);
}
void loop(){}
```

NOTE: This is exactly same like the regular Variable Declaration. Usage of the Variable from this Declaration is also the same. The difference is, with the "volatile" Keyword, we are asking the "Compiler Software" not to do any optimizing on the Variable

Optimizing activity normally get rid of "useless" codes, like the one in this example.

In this example PROGRAM: The optimizer decides that the code "*ptr=20" is "useless", therefore it is "removed". By putting a "volatile" Keyword on Variable "data" the optimizer will not touch "*ptr=20" because *ptr is related to "data" variable. This PROGRAM is just to demonstrate what the Optimizer can do, there is no reason for us to code like the above

However there are a few place where the Optimizer can interfere, thats where we use our "volatile" Keyword
- global variables that will be modified by interrupt
- global variables used in RTOS
- Variables for registers with pointers access

C-Langugae Keyword
17.volatile (Prevent the Compiler from automatic Optimizing of MEMORY)

ATMEGA328/Arduino Uno – MEMORY – VARIABLE – PART 3
https://github.com/teaksoon/lmaewapm

BASIC DATATYPE: Create/Declare Variable with "register" Keyword
Part1:"register" Keyword
Part2:name
Part3:value

Part1:"register" Keyword followed by space
Part2:datatype, followed by space
Part3:name, followed by a semi-colon ;

```
register datatype name;
```

Or with an initial value

```
register datatype name = 0;
```

Example PROGRAM:

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

```
register int counter = 0;

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

  Serial.print("\n\nData stored in variable counter = ");
  Serial.print(counter);
  Serial.print("\nNumber of BITS used by counter = ");
  Serial.print(sizeof(counter)*8);
}
void loop(){}
```

NOTE: This is exactly same like the regular Variable Declaration. Usage of the Variable from this Declaration is also the same.

The difference is, with the "register" Keyword, we are asking the "Compiler Software" to use the CPU Memory instead of the SRAM Memory. CPU Memory has higher speed than SRAM Memory.

Normally, the "Compiler Software" will already do it for us. So this "register" Keyword is not really needed

C-Language Keyword
18.register (Use CPU MEMORY instead of WORKING MEMORY) *compiler decides