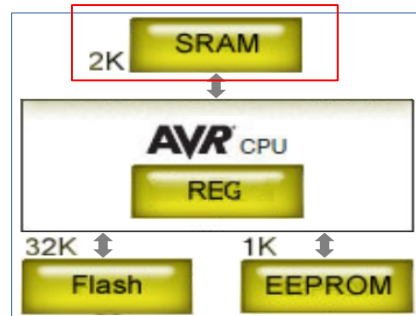


**WORKING MEMORY**

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		Symbols			
MEMORY	CONTROL	CONTROL	LOGIC	MATH	BIT OP
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		<b>MEMORY</b>			
19.typedef		&			
20.sizeof		*			

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 1 )

We will start with the most commonly used ones  
( this may be all that we need )

#### 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 BASIC Variable****Part1:**datatype**Part2:**name**Part1:**datatype, followed by space**Part2:**name, followed by a semi-colon ;**datatype name;****BASIC DATATYPE: Create/Declare BASIC Variable with initial value****Part1:**datatype**Part2:**name**Part3:**value**Part1:**datatype, followed by space**Part2:**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: THIS IS THE MOST COMMONLY USED CODE.** Variables declared with BASIC DATATYPE are used to store a single number

The most important thing about the DATATYPE is the size(number of BITS use). More BITS can store bigger number range, however MEMORY will be needed. We need to choose the DATATYPE wisely

**C-Language Keywords****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

- char,int,long,short stores whole numbers
- float,double stores numbers with fractions

**BASIC DATATYPE: Create/Declare Variable with "unsigned" Keyword****Part1:**"unsigned" Keyword**Part2:**datatype**Part3:**name**Part1:**"unsigned" Keyword followed by space**Part2:**datatype, followed by space**Part3:**name, followed by a semi-colon ;**unsigned datatype name;**

Or with an initial value

**unsigned datatype name = value;****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; // Positive 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:** This is like the normal BASIC DATATYPE where the Variable stores a single number.

- When the **"unsigned" Keyword IS NOT specified**, it is considered as **"signed"** (we can put the **"signed"** Keyword in front of our DATATYPE if we wish, however it is optional), The Variable can store both Positive and Negative number
- When the **"unsigned" Keyword IS specified**, the Variable can store Positive number only. If a negative number is assigned to a **"unsigned"** Variable, the number will be automatically converted into a positive number using the following two formats,
  - **"twos complement"** format for char,int,long,short (whole numbers)
  - **"IEEE 754"** format for float,double (numbers with fractions)

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

**BASIC DATATYPE: Create/Declare Variable with "const" Keyword**

**Part1:**"const" Keyword

**Part2:**datatype

**Part3:**name

**Part4:**value

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

**Part2:**datatype, followed by space

**Part3:**name, followed by an equal sign =

**Part4:**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 is "read-only" - CANNOT BE CHANGED

**C-Language Keywords**

**13.const** (Variable with Initial Data assigned that cannot be changed)

**BASIC DATATYPE: Create/Declare Variable with "static" Keyword****Part1:**"static" Keyword**Part2:**datatype**Part3:**name**Part4:**value**Part1:**"static" Keyword followed by space**Part2:**datatype, followed by space**Part3:**name, followed by an equal sign =**Part4:**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; // this is run just once
    counter = counter+1;
    return counter;
}

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

```

**NOTE:** We need to specify an initial Value for "static" Variable. When used in a function, the initial value will only be set once. Subsequence call to this function will not set initial value for the Variable again, the function will continue with whatever value that is already in the Variable

**C-Language Keywords****17.static** (Variable with Initial Data assigned just once, retain value)

**BASIC DATATYPE:Using "void" Keyword for function Declaration and Variable****Part1:**"void" Keyword**Part2:**name**Part1:**"void" Keyword followed by space**Part2:**name - Function name declaration**void name() {}**

Function with no return value

**void name(void) {}**

Function with no parameters (instead of "void" can also leave blank)

**void \*name;**

Memory Pointer Variable for unknown DATATYPE

**Part2:**name - Variable name declaration**Example PROGRAM:**Arduino IDE|Save PROGRAM as: **c\_variable\_void**

Enter codes below and upload. Use the Serial Monitor to see results

```

void setup(){
int    a=5;
float  b=5.67;
void  *ptr; // memory pointer Variable for unknown datatype

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

    // calling return_void() function, returns nothing
    return_void();
    // calling no_parameter() function, with no parameters
    no_parameter();

    ptr=&a; // Assigning address of integer to void pointer
    Serial.print("\nData stored in memory ptr = ");
    Serial.print( *((int*)ptr) );

    ptr=&b; // Assigning address of float to void pointer
    Serial.print("\nData stored in memory ptr = ");
    Serial.print( *((float*)ptr) );
}
void loop(){}

void return_void() {
    // do something here
    // this function does not return anything
}

void no_parameter(void) {
    // function oes not have any parameters
    // we can specify void or just leave the bracket ( ) pair blank
}

```

**NOTE:** void is an empty DATATYPE. There are many ways to use it, following are three ways of using it,

1. Use when function does not return anything
2. Use when function does not need any parameters (this is optional)
3. Use as Memory Pointer Variable to unknown DATATYPE

**C-Language Keywords****01.void** (Empty DATATYPE)