AVR
C Programming Language

Bits & Bytes

00000000 = 0
00000001 = 1
00000010 = 2
00000011 = 3
00000100 = 4
00000101 = 5
00000110 = 6
00000111 = 7
00001000 = 8

(9 thru 247)

1111000 = 248
1111001 = 249
1111010 = 250
1111011 = 251
1111100 = 252
1111101 = 253
1111110 = 254
1111111 = 255

00000001 = 0x01 = 1
00000010 = 0x02 = 2
00000100 = 0x04 = 4
00001000 = 0x08 = 8
00010000 = 0x10 = 16
00100000 = 0x20 = 32
01000000 = 0x40 = 64
10000000 = 0x80 = 128

Hexadecimal

0 = 0000 = 0x0
1 = 0001 = 0x1
2 = 0010 = 0x2
3 = 0011 = 0x3
4 = 0100 = 0x4
5 = 0101 = 0x5
6 = 0110 = 0x6
7 = 0111 = 0x7
8 = 1000 = 0x8
9 = 1001 = 0x9
10 = 1010 = 0xA
11 = 1011= 0xB
12 = 1100 = 0xC
13 = 1101 = 0xD
14 = 1110 = 0xE
15 = 1111 = 0xF

char

- The name of this data type is short for character, and is typically used to represent a character in the ASCII character set.
- A char is an 8-bit byte which can have 256 bit states. The computer uses this byte of data as representing a signed value for -128 to +127.

char

Table 9: ASCII Table

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
(nul)	0	0x00	(sp)	32	0x20	@	64	0x40	`	96	0x60
(soh)	1	0x01	!	33	0x21	A	65	0x41	a	97	0x61
(stx)	2	0x02	"	34	0x22	B	66	0x42	b	98	0x62
(etx)	3	0x03	#	35	0x23	C	67	0x43	c	99	0x63
(eot)	4	0x04	$	36	0x24	D	68	0x44	d	100	0x64
(enq)	5	0x05	%	37	0x25	E	69	0x45	e	101	0x65
(ack)	6	0x06	&	38	0x26	F	70	0x46	f	102	0x66
(bel)	7	0x07	'	39	0x27	G	71	0x47	g	103	0x67
(bs)	8	0x08	(	40	0x28	H	72	0x48	h	104	0x68
(ht)	9	0x09	)	41	0x29	I	73	0x49	i	105	0x69
(nl)	10	0x0a	*	42	0x2a	J	74	0x4a	j	106	0x6a
(vt)	11	0x0b	+	43	0x2b	K	75	0x4b	k	107	0x6b
(np)	12	0x0c	,	44	0x2c	L	76	0x4c	l	108	0x6c
(cr)	13	0x0d	-	45	0x2d	M	77	0x4d	m	109	0x6d
(so)	14	0x0e	.	46	0x2e	N	78	0x4e	n	110	0x6e
(si)	15	0x0f	/	47	0x2f	O	79	0x4f	o	111	0x6f
(dle)	16	0x10	0	48	0x30	P	80	0x50	p	112	0x70
(dc1)	17	0x11	1	49	0x31	Q	81	0x51	q	113	0x71
(dc2)	18	0x12	2	50	0x32	R	82	0x52	r	114	0x72
(dc3)	19	0x13	3	51	0x33	S	83	0x53	s	115	0x73
(dc4)	20	0x14	4	52	0x34	T	84	0x54	t	116	0x74
(nak)	21	0x15	5	53	0x35	U	85	0x55	u	117	0x75
(syn)	22	0x16	6	54	0x36	V	86	0x56	v	118	0x76
(etb)	23	0x17	7	55	0x37	W	87	0x57	w	119	0x77
(can)	24	0x18	8	56	0x38	X	88	0x58	x	120	0x78
(em)	25	0x19	9	57	0x39	Y	89	0x59	y	121	0x79
(sub)	26	0x1a	:	58	0x3a	Z	90	0x5a	z	122	0x7a
(esc)	27	0x1b	;	59	0x3b	[	91	0x5b	{	123	0x7b
(fs)	28	0x1c	<	60	0x3c	\	92	0x5c	|	124	0x7c
(gs)	29	0x1d	=	61	0x3d	]	93	0x5d	}	125	0x7d
(rs)	30	0x1e	>	62	0x3e	^	94	0x5e	~	126	0x7e
(us)	31	0x1f	?	63	0x3f	_	95	0x5f	(del)	127	0x7f

unsigned

- If the modifier unsigned is used in the definition of a char variable: ‘unsigned char’, the value is from 0 to 255.
- Byte, byte: unsigned char

int

- On AVR microcontrollers int declares a 16 bit(2 bytes) data variable as having values from -32768 to +32767. A variable declared with ‘unsigned int’ will have a value from 0 to 65535.
- long int, short int: machine dependent. short int: 16 bits. long int: 32 bits.

Arithmetic Operators

Operator	Name	Example	Defined
*	Multiplication	x*y	Multiply x times y
/	Division	x/y	Divide x by y
%	Modulo	x%y	Provide the remainder of x divided by y
+	Addition	x+y	Add x and y
-	Subtraction	x-y	Subtract y from x
++	Increment	x++	Increment x after using it
--	Decrement	--x	Decrement x before using it
-	Negation	-x	Multiply x by –1
+	Unary Plus	+x	Show x is positive (not really needed)

Data Access and Size Operators

Operator	Name	Example	Defined
[]	Array element	x[6]	Seventh element of array x
.	Member selection	PORTD.2	Bit 2 of Port D
->	Member selection	pStruct->x	Member x of the structure pointed to by pStruct
*	Indirection	*p	Contents of memory located at address p
&	Address of	&x	Address of the variable x

Structure Pointer Example

```
#include <stdio.h>
struct person
{
   int age;
   float weight;
};

int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;
    printf("Enter age:");
    scanf("%d", &personPtr->age);
    printf("Enter weight:");
    scanf("%f", &personPtr->weight);
    printf("Displaying:\n");
    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);
    return 0;
}
```

Logical and Relational Operators

Operator	Name	Example	Defined
>	Greater than	x>y	1 if x is greater than y, otherwise 0
>=	Greater than or equal to	x>=y	1 if x is greater than or equal to y, otherwise 0
<	Less than	x<y	1 if x is less than y, otherwise 0
<=	Less than or equal to	x<=y	1 if x is less than or equal to y, otherwise 0
==	Equal to	x==y	1 if x equals y, otherwise 0
!=	Not equal to	x!=y	1 if x is not equal to y, otherwise 0
!	Logical NOT	!x	1 if x is 0, otherwise 0
&&	Logical AND	x&&y	0 if either x or y is 0, otherwise 1
||	Logical OR	x||y	0 if both x and y are 0, otherwise 1

Bitwise Operators

Operator	Name	Example	Defined
~	Bitwise complement NOT	~x	Changes 1 bits to 0 and 0 bits to 1
&	Bitwise AND	x&y	Bitwise AND of x and y
|	Bitwise OR	x|y	Bitwise OR of x and y
^	Bitwise exclusive OR	x^y	Bitwise XOR of x and y
<<	Left shift	x<<2	Bits in x shifted left 2 bit positions
>>	Right shift	x>>3	Bits in x shifted right 3 bit positions

Bitwise Operators

```
myByte = 11111111 = 0xFF
  0x08 = 00001000 = 0x00
-----------------------
    OR = 11111111 = 0xFF

myByte = 01010101 = 0x55
  0x08 = 00001000 = 0x08
-----------------------
   AND = 00000000 = 0x00
```

```
myByte = 01010101 = 0x55
  0x08 = 00001000 = 0x08
------------------
    OR = 01011101 = 0x5D

myByte = 10101011 = 0xAA
  0x08 = 00001000 = 0x08
------------------------
   AND = 00001000 = 0x08
```

```
 0x20 = 00100000
~0x20 = 11011111
```

Assignment Operators and Expressions

Operator	Name	Example	Defined
=	Assignment	x=y	Put the value of y into x
+= -= *= /= %= <<= >>= &= ^= |=	Compound assignment	x += y	This provides a short cut way to write and expression, the example: x += y; is the same as x = x + y;

Conditional Expressions

```
if( temp > 150)
        Fan(ON);
else
        Fan(OFF);
```

```
temp > 150 ? Fan(ON) : Fan(OFF);
```

- The operation has the form: expression1 ? expression2 : expression 3, and follows the rule that if expression1 is true (non-zero value) then use expression2, otherwise use expression3.

If-Else and Else-If

```
if (expression)
      statement1
else
      statement2
```

```
if (expression1)
      statement1
else if (expression2)
      statement2
else if (expression3)
      statement3
else
      statement4
```

```
if(input == KEY_PLUS)PORTD= ~0x01;
else if(input == KEY_NEXT)PORTD = ~0x02;
else if(input == KEY_PREV)PORTD = ~0x04;
else if(input == KEY_MINUS)PORTD = ~0x08;
else if(input == KEY_ENTER)PORTD = ~0x10;
```

Switch

```
switch (expression) {
      case constant expression1 : statements
      case constant expression2 : statements
      case constant expression31 : statements
      default: statements
}
```

```
switch(input){
      case KEY_PLUS :
            PORTD = ~0x01;
            break:
      case KEY_NEXT :
            PORTD = ~0x02;
            break;
      case KEY_PREV :
            PORTD = ~0x04;
            break;
      case KEY_MINUS :
            PORTD = ~0x08;
            break;
      case KEY_ENTER :
            PORTD = ~0x10;
            break;
      default:
}
```

Switch

```
switch( input){
      case 'a' :
      case 'A' :
            DoaA();
            break;
      case 'b' :
      case 'B' :
            DobB();
            break;
      case '0' :
      case '1' :
      case '2' :
      case '3' :
            Gofer0123();
            break;
      case '4' :
      case '5' :
      case '6' :
      case '7' :
            Gofer4567();
            break;
      default:
            DoDefault();
            break;
}
```

```
switch( input){

      case 'a' : case 'A' :
            DoaA();
            break;

      case 'b' : case 'B' :
            DobB();
            break;
      case '0' : case '1' : case '2' : case '3' :
            Gofer0123();
            break;
      case '4' : case '5' : case '6' : case '7' :
            Gofer4567();
            break;
      default:
            DoDefault();
            break;
}
```

Loops

```
while(expression)
{
      // Do stuff while expression is true
}
```

```
xint i;
while( i <= 128)
{
      PORTD = i;
      _delay_loop_2(30000);
      i = i*2;
}
```

For loop

```
for(expresson1; expression2; expresson3)
{
        // Do stuff
}
```

```
for(int i = 1; i <= 128; i = i*2)
{
        PORTD = i;
        _delay_loop_2(30000);
}
```

For loop, do-while loop

```
for(;;)
{
        // Do stuff forever
}
```

```
do
{
        // Do stuff at least once
}
while(expression);
```

Functions

```
char adder(unsigned char ad1, unsigned char a1)
{
      unsigned char r;

      r = a1 + a2;

      if(r == 2) getrewarded();
      else getboinked();

      return r;
}


int main()
{
      unsigned char add1 = 1;
      unsigned char add2 = 1;
      unsigned char results = 0;

      results = adder(add1,add2);

      if(results == 2) getrewarded();
      else getboinked();
}
```

functions

```
void adder(unsigned char, unsigned char);
unsigned char results = 0;

int main()
{
      unsigned char add1 = 1;
      unsigned char add2 = 1;

      adder(add1,add2);

      if(results == 2) getrewarded();
      else getboinked();
}

void adder(unsigned char ad1, unsigned char a1)
{
      results = a1 + a2;
}
```

extern

In file1:

```
extern double gadabout;
extern char harlot;
```

In file2:

```
double gadabout = 0;
char harlot = '?';
```

Headers

Header files are a convenient place to stick all the stuff that you put before the main() function. They are files with a suffix of .h and are declared as:

```
#include <LEDblinker.h>
#include "PCcomm.h"
```

If the declaration uses <filename> the compiler looks in an implementation defined location, usually an 'include' directory. If it uses "filename" the compiler looks in the same directory that the source program was located. The choice will depend on how you've decided to organize your development file system.

Macro Substition

We can use #define to make a simple token that replaces a complex, or frequently used expression. For example we may want to determine the larger of two variables:

```
#define larger( x, y)   ( (x)>(y) ? (x) : (y) )
```

Which we would use as:

```
int a= 9;
int b = 7;
int c = 0;

c = larger( a, b);
```

The preprocessor replaces the last statement with:

```
c = ( (a)>(b) ? (a) : (b) );
```

Which is what the compiler sees.

Pointers

```
#include <stdio.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = var;
   for ( i = 0; i < MAX; i++)
   {

      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* move to the next location */
      ptr++;
   }
   return 0;
}
```

```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200
```

Array of Pointers

```
#include <stdio.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int i, *ptr[MAX];

   for ( i = 0; i < MAX; i++)
   {
      ptr[i] = &var[i]; /* assign the address of integer. */
   }
   for ( i = 0; i < MAX; i++)
   {
      printf("Value of var[%d] = %d\n", i, *ptr[i] );
   }
   return 0;
}
```

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

Passing Pointers to Functions

```
#include <stdio.h>
#include <time.h>

void getSeconds(unsigned long *par);

int main ()
{
   unsigned long sec;


   getSeconds( &sec );

   /* print the actual value */
   printf("Number of seconds: %ld\n", sec );

   return 0;
}

void getSeconds(unsigned long *par)
{
   /* get the current number of seconds */
   *par = time( NULL );
   return;
}
```

```c
#include <stdio.h>

/* function declaration */
double getAverage(int *arr, int size);

int main ()
{
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value  */
   printf("Average value is: %f\n", avg );

   return 0;
}
```

```
double getAverage(int *arr, int size)
{
  int    i, sum = 0;
  double avg;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }

  avg = (double)sum / size;

  return avg;
}
```

```
Average value is: 214.40000
```

Strings

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

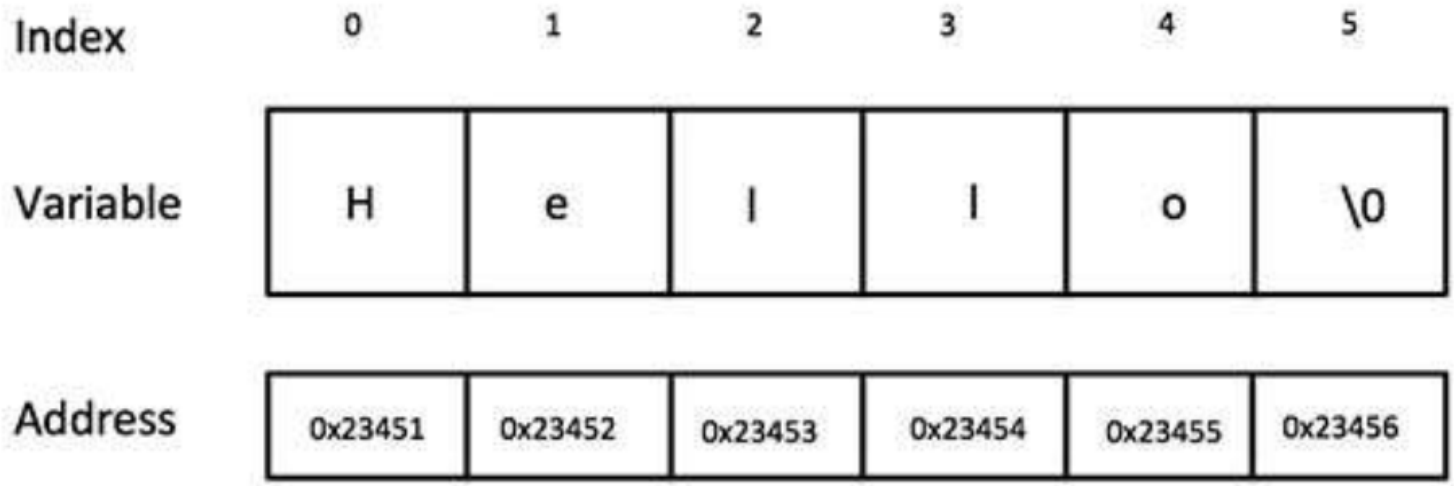
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization, then you can write the above statement as follows:

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C/C++:



Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string:

```
#include <stdio.h>


int main ()
{
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};


   printf("Greeting message: %s\n", greeting );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Greeting message: Hello
```

String Functions

S.N.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

```c
#include <stdio.h>
#include <string.h>

int main ()
{
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;

   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );

   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );
```

```
   /* total lenghth of str1 after concatenation */
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

Structures

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type. The following example shows how to use a structure in a program:

```
#include <stdio.h>
#include <string.h>

struct Books
{
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};
```

```c
int main( )
{
   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
```

```
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);

   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 subject : %s\n", Book2.subject);
   printf( "Book 2 book_id : %d\n", Book2.book_id);

   return 0;
}
```

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```