BIG DADDY'S

Basic C for Beginners

C was designed as a fast, efficient, compiled, low level language, with a minimum of overhead. (Many other languages were written in C.) The Arduino is programmed in a modified subset of C++, a superset of C. The 4 fundamental tasks in creating a C program are; **Edit** (create source code), **Compile** (2 stage process), **Link** (get dependent functions), **Execute**. This toolbox uses compact bracket formating.

```
char hi[ ] = "Hello!";
printf("%s\n", hi);
```

empty [ ] lets compiler assign length to string array automatically

Typical C Program Components Structure

HEADER SEGMENTS

> **Documentation** /* global comments enclosed */

> **Preprocessor statements** - *no ; at statement end*

- System header files **#include <file>**
  ex: #include <stdio.h> (std path) or **"mylink"**
  also: **#define #if #ifdef #ifndef #else #elif #endif**
  preprocessor looks in source file directory
- **Constants** **#define NAME value** (note caps)
  ex: #define MYPI 3.1416

> **Function prototype declarations** <-alerts compiler
return-type name required-parameter-type
ex: int mySub(int) <-full declaration later

> **Global Variables**

BODY *return-type main()* <-Required in body of
ex: int main(void) every **C** program
{ <- opening main() bracket

- local variable definitions;
- program statements;
- /* local documentation */

program

} <-after the return statement

RETURN *return 0;* **or** *return;*

> followed by closing main() bracket }

FUNCTIONS
Programmer's functions neatly tucked down here to keep them out of main program flow

Some Data Types\Format Specifiers

(find exact size of a type with **sizeof** operator)

int	integer	%d or %i	preceed by 0x for a hexa-decimal value
(short)	2 bytes: -32,768 to 32,767		
	4 bytes: +/- ~ 2,147,483,647		
		%x	hexadecimal
char	character	%c	1 char, in single ' , 'A'
	string	%s	chars in double ", "ABC"
flopat	decimal	%f	123.456
double	decimal	%lf	double precision f
long long int	long long int	%li	usually 8 bytes
_Bool	boolean	%d	holds 1 or 0
size-t	unsigned int		use for large array index
void	absense of a type, no value available		
pointer		%p	
enum	programmer defined: specifies valid values		

ex: enum lightColors {red=1, yellow, green};
enum lightColors stop=red, warn=yellow, go=green;
printf(" %d, %d, %d", stop, go, warn); yields 1, 3, 2

Pointers

C uses pointers extensively and can-not be used functionally without them ***requires #include <stddef.h>***

***** - creates a pointer
***** - "dereferences", i.e., gets the pointed to value
& - gets address a pointer will hold
Create a pointer with a NULL starting value:
*type * name = NULL* ex: int * pMyVal = NULL;
Assign/Initalize pointer: **type * pName = &Variable**
Assign the address of a variable to be the value of the pointer: **= &variable** ex: pMyPtr = &myVal
Assign new value to the address held by a pointer:
***pointer = newValue**
To **dereference** a pointer, i.e. get the **value held** at the **address held** by the pointer: **= *pointer**
ex: int myNewVal = 0; myNewVal = *pMyVal;
Print the address of a pointer (pMyVal below): ex:
printf("pMyVal address: %p\n", (void*)&pMyVal);

Control Structures

Comparison: IF
if (condition) {statements}
else if (condition) {statements}
else {statements}

Comparison to Constant Values: SWITCH

```
switch (expression)    {
    case value1:
    program statements;
    break; / continue; / exit(x);
    case value2:
    program statements;
    break; / continue; / exit(x);
    default:
    program statements;          }
```

Loops: For, While, Do-while

For: Counter Loop - Loop utill a count is reached
~ create counter variable first (ex: **int i;**)
~ counter initialized before loop, tested at END of the loop
for (initialization; counter limit condition; step expression)
{ statements; } ex: **for(i = 1; i < 11; ++i) {statements;}**

For: Condition Sentinel - loop/execute till condition is met
~ condition tested at the **start** of the loop
for ([variables values initialization]; true continuation con-dition, action per iteration) { statements; }

For: Infinite loop - uses break statement in process to exit
~ no condition ever tested by for statement
for(;;) { code; } or equivalently while(1) { code; }

While single statement
while (exit condition expression) statement; assume test is an int variable = 0: while(++test < 5) printf("%d\n",test);

While multiple statements
while(exit condition expression) { statements; }

Do-While (always executed at least once)
do {
statements; }
while (expression is true);

Bifurcation Statements:
break; exits a loop
exit(); exits program
goto label; *shifts execu-tion to line*
label:; *at label:*
continue; ends iteration, continues with next iteration

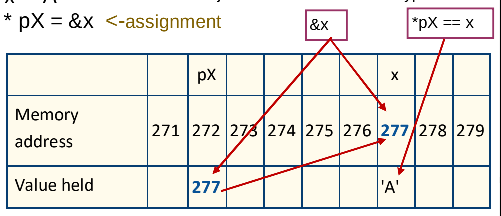
The Tokens of C

Constants (Literals) Identifiers (main, ...); Keywords (for, int, if, while..); Strings; Operators (*, -, *, =, ==, >= ...); Special Symobols ([,],{,},(,),\ ...)

How Pointers Work

char x = 'A'
char * pX = &x <-assignment

Pointer Arithmetic: +.-.++.-- compiler adjusts address for variable type



Types of Variables

local: defined inside a function/ block
global: defined outside all functions
formal: defined in a parameter of a function - treated as local to the func-tion, takes precedence over globals

Constants (Literals)

'a',0,3.17,10 to preprocess:
#define NAME amount
use all CAPS, no semi-colon.
In main() body, const keyword creates immutable value OR pointer location: char const *
arrayEnd = array + n *where n is array len fixes last arrary addr.*

Arrays

type ArrayName [# elements] [ ]
sequential, **starts with element [0],** can be multidimensional, initialization is important!
Compiler does NOT check for out-of-bounds errors! ex:
int ary[3][2] = {{1,2} {3,4} {5,6}}
can use variables to define but can not initialize a Variable Length Array (VLA) when created

Strings

***char name* = 'character'** name = 'a'; defines a variable of a single character. *Note single quotes.*
char name[#] = " char string"
creates a string array with a maximum of # characters, starts at [0], leave # blank and compiler will assign legth of initialzed string adding 1 for a terminating char-acter of '\0'. Can not compare strings with ==. Can not assign one string to another unless calling strncpy(). See <string.h>

C Keywords

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

↓ **Note:** there is **no** thousands grouping flag in C!

Format / Specifiers ⇒

%[parameter] [flags] [width] [.precision] [length] type

- [parameter]: **n$** is the # of the parameter to display if multiple parameter outputs
- [width]: minimum char-acters output
- [.precision]: specifies maximum limit on ouput - # of digits to right of decimal
- [length]: omit, or hh (chr - int), h, l, ll, L (long dbl), z (size_t) , j, t
- [flags]: "-" left-align; "+" prepends sign; " " (space) pre-pends space for + signed, - negative; 0 (zero) if width is spec'ed, prepends zeros for numerics; # alternate forms
- type: %: **d** *signed int*, **u** *unsigned int + only*, **f** *double fixed*, **e** *double scientific*, **g** *double format by size*, **o** *unsigned octal*, **s** *null ending string*, **c** *char*, **p** *void **, **a** *double hex*, **n** *print nothing*

BIG DADDY'S

Basic C for Beginners

R1

v070120a

comments / suggestions appreciated:
john@johnoakey.com

Structures: element groups, no memory allocated

To Create a struct:
struct-keyword   name-of-this-struct
{ struct   date
  variable definitions
};

int month;
int day;
int year;

To Create an instance
struct, name, instanceName; {vars};
struct date today; {today.month=11;…}
Reference a field   *use dot operator*
today.year=2020   *with no space*
Define pointer variable to a struct
struct name pointer-name
struct dates *datesPtr;
To assign variable value to pointer
pointer-name = &instance-name
datesPtr = &today
IMPORTANT NOTE: Since structs allocate no space, string arrays given pointers must have already been defined or had space "malloced"!
pointers can access a field of an instance   (*datesPtr).day = 19
(*parens req by precedence of dot op*)
A special operator (->) derefs and selects instance of a field at once:
if(datesPtr->month == 12) is same as
if((*datePtr).month ==12)
Structures containing pointers: same rules apply. *To assign ptr values*
instance.pointer_variable= &variable
or *pointer_variable = real#/constant
*Create an **array of structures***
struct dates myDates[10] *set values:*
myDates[3].year=1948; *and/or*
struct dates myDates[9]={{12,24,1948},{1,19,1948},{3,2,1970}}; *sets 0,1,2 of 9*
*Create structures **containing** arrays*
struct struct-name { variable definitions including arrays }
To access and set array elements
instance_name.element[#] = value
aMonth.name[0]= 'J'   etc., or
struct month aMonth={{'J','a','n'}[other] };
Nested Structures: can create a struct to hold other structs
struct dateAndTime {struct date sdate;
struct time stime;}; *binds sdate & stime*
To Create a nested instance
struct structure-name instance-name;
struct dateAndTime **event**;
To access / change event element
event.**sdate**.month=12;
++event.stime.seconds; *(<adds a sec)*
Structures and Functions
assuming typical struct: define func:
return_type func_name (**struct** instancea, **struct** instanceb …) **{**
code; return; **}**   should always use ptrs if passing struct to funct ↓
Struct Pointers as function arguments
avoid memory use and cpu time:
return_type func_ name (struct *instancea, struct *instanceb…)**{...**
reference pointerToInstance->field
use *const before struct pointer name to stop data changing:; after locks address
prototype to return a struct
struct   struct_name   func_name(void)

Reading\Writing to a File on Disk

#include <stdio.h> attaches in/out functions
can read/write text or binary files; TEXT operations:
note: **EOF** = end of file;   assumption: "file" is in current dir;
FILE * (or fp) is "file pointer", creates a pointer to file name
For a file "Mary.txt"   char * Mary = "Mary.txt";
Create an uninitialized pointer variable FILE *pMary = NULL; *Initialize file pointer* pMary= fopen("name" / [pointer], "mode") pMary = fopen(Mary,"w+");   **or**
pMary = fopen("Mary.txt", "w+"); <- argument 2 ("w+")
opens (associates or initializes) a file for access **type**
Modes: **"w"** - creates (overwrites) file for writing, **"a"** - append (create if new), **"r"** - opens to read, **"w+"** - creates to write & read, **"a+"** - opens to read and append, **"r+"** - opens file to read or write
~ **must** test successful opening of file after fopen():

```
  if(pName == NULL)  {
       printf("Failed to open %s\n", fileNameVar);  }
```

fclose(pName) - closes file; success returns int 0
rewind(pName) - reset pointer to start of file
rename(pOld, pNew**)** - renames; 0 ret'd if successful
int rename(*oldName, *newName);
~ example with absolute path:
if (rename("C:\\temp\\myfile.txt", C:\\temp\\myfile_copy.txt"))
remove("myfile.txt"**)** - deletes myfile in current dir.

Reading from a text file:

fgetc() - 1 char, then advances position indicator, can be a macro, EOF at end, **int** xchr=**fgetc(fp)**
after initialization command is just xchr=fgetc(fp)
~~getc()~~ **use fgetc** int xchr=**getc(fp)**; gets 1 char
fgets() - reads stream to first \n or #chars into *str
fgets(pointer_to_array_to_hold_str_read, (int) #chars to read, fp stream)
scanf() - reads formatted data from stdin; scanf (format, str array); char xary[25]; scanf(%s, xary);
formats: [* ignore], [width max], [modifiers], type=;
(types include %c, %d, %f, %o, %s, %u, %x, … etc)
fscanf() - fscanf(fp, "data_format(s)",vars);
fscanf(fp, "%s %d %s", sary1, myint, sary2);

Writing to a text file:

puts(char array pointer) - prints char string in array
fputc() - fputc(int char, fp); fputc(33, pMary);
fputs() - writes stream; fputs("text \n", fp)
fprintf() - write formatted data; fprintf(fp, format(s), variables); fprintf(fp, "%s %d %s", "at", 12, "pm");

File Positioning for Access:

fpos_t stores current file position: fpos_t here;
ftell(FILE *) takes file ptr, returns position (long int) as offset to start of file; long fpos = ftell(fp)
fseek(fp, offset, int ref point) - offset is from ref, remember EOF; ref point is one of: **SEEK_SET** (start of file) *or **SEEK_CUR** (binary files), **SEEK_END*** *(EOF)*
fseek(fp,0,SEEK_END) sets file pos at EOF, so len = ftell(fp) will yield the length of the file in var len
fgetpos(FILE* fp, &position); fgetpos(fp, &pos)
fsetpos(FILE* fp, fpos_t *pos); fsetpos(fp, &pos);

Operators by Priority

			Bitwise	*= mult/asgn
			& AND	/= div asgn
:: scope	! unary NOT	- subtraction	^ NOT	%= mod asgn
() parens	& ptr ref	<< bit left	| OR	
[ ] brackets	* ptr deref	>> bit right		Bitwise
-> point ref	(type) cast	< less than	&& locigal and	>> shft/asgn
. struct ele	+- unary less	<= less/equal	|| logical OR	<< shft/asgn
sizeof mem	* multiply	> more	?: conditional	&= AND asgn
++ increment	/ divide	>= more/equal	= assignment	^= NOT asgn
-- decrement	% modulus	== same as	+= add/asgn	|= OR asgn
~ bitw compl	+ addition	!= not equal	-= sub/asgn	, comma

Functions (procedure, subroutine, module)

Built in: See keywords
Standard: Standard library provides many functions in header files with #include <file_name> statement
User Defined:
Declaration: header statement before main() which tells the compiler there is a local in-line function and specifies: **return-type name ([parameters]);** int myfun(int *num) *or* char mySub(int) Parameters are values passed to the function which may or may not return a value, if not it is type void.
Definition: The actual body of the function - placed above or below main() and has the syntax:

```
return-type  name  ([parameters])       {
    code
    return;  or return(value);        }
```

Argument Call types: (call by value is default)
Call **by Value**: copies value of argument to function parameter - does not effect the actual argument.
Call **by Reference**: copies address of argument to function - changing value using the address pointer **does** change the original argument.
Calling: a function is called by coding its name as a statement ex: aTest(); or by using it to assign a value to a variable - ex: int myint = aTest(mychar);

A Few Select Essential Functions Available in <header files>

/* comments */

<stdio.h>
size_t, FILE, fos_t
NULL, EOF, SEEK_CUR, SEEK_END, SEEK_SET, stderr, stdin, stdout
getchar(void);
printf(const char *, ...);
gets(char *);
putc(int, FILE *);

<stdlib.h>
size_t, NULL
calloc(size_t nitems, size_t size)
free(void *ptr)
malloc(size_t size)
realloc(void *ptr, size_t size)
exit(int status)
abs(int x)
div(int numer, int denom)
rand(void)
srand(unsigned int seed)
atof(const char *str)
atoi(const char *str)
atol(const char *str)
strtod(con char *str, char **endptr)
strtol(con ch *str, ch **endp, int...
strtoul(con ch *str, ch **endp, in...
abort(void)
labs(long int x)
ldiv(long int numer, long int denom)

<string.h>
size_t, NULL
strcat(chr *dest, const chr *src)
strncat(chr *dest, con chr * ...
strchr(const char *str, int c)
strcmp(const char *str1, con...
strncmp(con chr *str1, con ch...
strncpy(chr *dest, con chr *src...
strlen(const char *str)
strpbrk(con chr *st1, con chr*st2
strrchr(const char *str, int c)
strtok(chr *str, con char *delim)
strstr(con cr *haystack, *needle)
strxfrm(chr *dest, con chr *src...
memchr(con void *str, int c, si...
memcmp(con void *s1, con vo..
memcpy(void *dest, con void *..
memmove(void *dest, const void *src, size_t n)

<ctype.h>
all character classes
isalnum(int c) **isalpha**(int c)
iscntrl(int c) **isdigit**(int c)
isgraph(int c) **islower**(int c)
isprint(int c) **ispunct**(int c)
isspace(int c) **isupper**(int c)
tolower(int c) **toupper**(int c)

<math.h>
modf(double x, double *integer)
pow(double x, double y)
sqrt(double x)
ceil(double x)
fabs(double x)
floor(double x)
fmod(double x, double y)
acos(double x)
asin(double x)
atan(double x)
atan2(double y, double x)
cos(double x)
cosh(double x)
sin(double x)
sinh(double x)
tanh(double x)
exp(double x)
log(double x)
log10(double x)

<time.h>
size_t, clock_t (stores processor time), time_t (for calendar time),
struct tm a structure to hold the time and date: *tm_sec; tm_min; tm_hour; tm_mday; tm_mon; tm_year; tm_wday; tm_yday; tm_isdst*
char ***asctime**(...) day&time of ptr
clock_t **clock**(void) processor cyc
char ***ctime**(...) local time
double **difftime**(...) dif in secs
struct tm ***gmtime**(…) timer>GMT
struct tm ***localtime**(…) timer>local
size_t **strftime**(…) formatted time
time_t **time**(time_t *timer) cal time