ANSI LIBRARY : stdio, stdlib, string, math, ctype, time, stdarg, limits

The following is a synopsis of functions available in the standard ANSI C library. Other on-line reference tools (i.e., Symantec THINK Reference and/or Toolbox Assistant) provide a more detailed review of the ANSI C Library functions, but a quick review of these functions is provided in this section for your convenience.

As a note, if you plan on programming for the MacOS, you'll find that you won't be using the ANSI C library as much as you might think. This goes especially for input and output functions. This means that you probably don't need to spend a lot of time learning the ins and outs of the ANSI C library - although you should at least be aware of the capabilities it provides.

Here's a list of the ANSI C library header files:

<stdio.h>
<stdlib.h>
<string.h>
<math.h>
<ctype.h>
<time.h>
<stdarg.h>
<limits.h>
<float.h>
<assert.h>
<setjmp.h>
<signal.h>

STANDARD INPUT/OUTPUT <stdio.h>

Filenames are limited to FILENAME_MAX characters. At most FOPEN_MAX files may be open at a given time.

CLEARERR
void clearerr(FILE *stream)

FCLOSE
int fclose(FILE *stream)
Returns EOF if error occurs, zero otherwise.

FEOF
int feof(FILE *stream)
Returns non-zero if EOF is set.

FERROR
int ferror(FILE *stream)
Returns non-zero if error exists for 'stream'.

FFLUSH
int fflush(FILE *stream)
Returns EOF if error, zero otherwise. fflush(NULL) flushes all output streams.

FGETC
int fgetc(FILE *stream)
Returns EOF if error.

FGETS
char *fgets(char *s, int n, FILE *stream)
Returns s, or NULL if error.

FGETPOS
int fgetpos(FILE *stream, fpos_t *ptr)
Records current position in 'stream' in '*ptr'.

FOPEN
FILE *fopen(const char *filename, const char *mode)
modes: "r" read only
"w" write; discard previous contents
"a" append
"r+" read and write
"rb" read (binary)
"wb" write (binary)
"ab" append (binary)
"r+b" read and write (binary)

FPRINTF
int fprintf(FILE *stream, const char *format, ...)
Returns number of characters printed, or negative result if error. Conversion specification begins with '%' and ends with conversion character. In between, there may be (in order):
flags:
'-' left justify
'+' print with sign
' ' <space> prefix with space if no sign
'0' <zero> specifies padding with leading zeros
'#' alternate output form
numbers:
n first number specifies field width
'.' period used as separator
n second number specifies precision (max characters)
m modifier (h - short, l - long, L - long double)
format characters:
d,i int; signed
o int; unsigned octal (without leading zero)
x,X int; unsigned hexadecimal
u int; unsigned
c int; single character
s char *; string
f double;
e,E double; scientific notation
g,G double; %e or %f format, depending on value
p void *; pointer
n int *; number of characters written so far
% no argument, print '%'
example:
fprintf(myFilename, "The results are %d and %6.2f\n", myInt, myReal);

FPUTC
int fputc(int c, FILE *stream)
Returns c, or EOF if error.

FPUTS
int fputs(const char *s, FILE *stream)
Returns non-negative result, or EOF if error.

FREAD
size_t fread(void *ptr, size_t size, size_t nobj, FILE *stream)
Reads from 'stream' into 'ptr' at most 'nobj' objects of size 'size'. Returns number of objects read.

FREOPEN
FILE *freopen(const char *filename, const char *mode, FILE *stream)
Returns NULL if error.

FSCANF
int fscanf(FILE *stream, const char *format, ...)
Reads from 'stream' and assigns values through subsequent arguments which must be pointers. Returns EOF if end of file occurs before any conversion; otherwise it returns number of items assigned. Format string should contain conversion specifications consisting of '%' followed by:
flag:
    * suppression; ignore field
number:
    n first number specifies maximum field width
width character:
    h short
    l long
    L long double
conversion characters:
    d int *
    i int *; may be octal (leading 0) or hexadecimal (leading 0x)
    o int *; octal integer
    x int *; hexadecimal integer
    c char *; characters
    s char *; string of non-white characters
  e,f float *; could also use 'g'
    p void *; pointer
    n int *; writes into arg number of characters read. Nothing read.

FSEEK
int fseek(FILE *stream, long offset, int origin)
Sets file position. Values for origin can be:
  SEEK_SET beginning
  SEEK_CUR current position
  SEEK_END end of file

FSETPOS
int fsetpos(FILE *stream, const fpos_t *ptr)
Positions 'stream' at position recorded by fgetpos. Returns non-zero if error.

FTELL
long ftell(FILE *stream)
Returns current position for 'stream', or -1L if error.

FWRITE
size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *stream)
Writes from 'ptr' to 'stream' 'nobj' objects of size 'size'. Returns number of objects read.

GETC
int getc(FILE *stream)
Same as fgetc, except a macro.

GETCHAR
int getchar(void)
Equivalent to getc(stdin).

GETS
char *gets(char *s)
Returns s, or NULL if error.

PERROR
void perror(const char *s)
Prints error message 's'.

PRINTF
int printf(const char *format, ...)
Equivalent to fprintf(stdout, ...)

PUTC
int putc(int c, FILE *stream)
Same as fputc, except a macro.

PUTCHAR
int putchar(int c)
Equivalent to putc(c, stdout).

PUTS
int puts(const char *s)
Returns non-negative result, or EOF if error.

REMOVE
int remove(cost char *filename)
Returns non-zero if attempt to delete file fails.

RENAME
int rename(const char *oldname, const char *newname)
Returns non-zero if attempt to rename file fails.

REWIND
void rewind(FILE *stream)

SCANF
int scanf(const char *format, ...)
Identical to fscanf(stdin, ...)

SPRINTF
int sprintf(char *s, const char *format, ...)
Same as printf except output written to string 's'.

SSCANF
int sscanf(char *s, const char *format, ...)
Equivalent to scanf(...) except input is taken from 's'.

SETBUF
void setbuf(FILE *stream, char *buf)
If 'buf' is NULL, buffering is turned off, otherwise equivalent to 'void setvbuf(stream, buf, _IOFBF, BUFSIZE)'.

SETVBUF
int setvbuf(FILE *stream, char *buf, int mode, size_t size)
Returns non-zero if error. A value of NULL for 'buf' allocates buffer.
modes: _IOFBF full buffering
_IOLBF line buffering
_IONBF no buffering

TMPFILE
FILE *tmpfile(void)
Returns NULL if error.

TMPNAM
char *tmpnam(char s[L_tmpnam])
tmpnam(NULL) creates unique filename. TMP_MAX different names guaranteed.

UNGETC
int ungetc(int c, FILE *stream)
pushes 'c' (converted to an unsigned char) back onto 'stream'. Returns 'c' or EOF if error.

VPRINTF
int vprintf(const char *format, va_list arg)

VFPRINTF
int vfprintf(FILE *stream, const char *format, va_list arg)

VSPRINTF
int vsprintf(char *s, const char *format, va_list arg)

STANDARD LIBRARY <stdlib.h>

ABORT
void abort(void)
Causes program to abnormally terminate.

ABS
int abs(int n)

ATEXIT
int atexit(void (*fcn)(void))
Registers function to be called when program terminates. Returns non-zero if error.

ATOF
double atof(const char *s)

ATOI
int atoi(const char *s)

ATOL
long atol(const char *s)

BSEARCH
void *bsearch(const void *key, const void *base, size_t n, size_t size, int (*cmp)(const void *keyval, const void *datum))

CALLOC
void *calloc(size_t nobj, size_t size)

DIV
div_t div(int num, int denom)

EXIT
void exit(int status)
Causes normal program termination.

FREE
void free(void *p)
Deallocates spaced pointed to by 'p'. Returns NULL if error.

GETENV
char *getenv(const char *name)

LABS
long labs(long n)

LDIV
ldiv_t ldiv(long num, long denom)

MALLOC
void *malloc(size_t size)
Returns pointer for object of size 'size', or NULL if request for memory denied.

QSORT
void qsort(void *base, size_t n, size_t size, int (*cmp)(const void *, const void *))

RAND
int rand(void)
Returns pseudo-random integer between 0 and RAND_MAX.

REALLOC
void *realloc(void *p, size_t size)

STRTOD
double strtod(const char *s, char **endp)

STRTOL
long strtol(const char *s, char **endp, int base)

STRTOUL
unsigned long strtoul(const char *s, char **endp, int base)

SRAND
void srand(unsigned int seed)

SYSTEM
int system(const char *s)

STRING FUNCTIONS <string.h>

MEMCHR
void *memchr(const char *cs, int c, size_t n)
Returns first occurrence of character 'c' in 'cs' or NULL.

MEMCMP
int memcmp(const char *cs, const char *ct, size_t n)
Compares 'n' characters of 'cs' with 'ct'.

MEMCPY
void *memcpy(char *s, const char *ct, size_t n)
Copy 'n' characters from 'ct' to 's', returns 's'.

MEMMOVE
void *memmove(char *s, const char *ct, size_t n)
Same as 'memcpy'. Works if objects overlap.

MEMSET
void *memset(char *s, int c, size_t n)
Put character 'c' into first 'n' characters of 's', returns 's'.

STRCAT
char *(char *char *s, const char *ct)
Concatenates 'ct' to 's', returns 's'.

STRCHR
char *strchr(const char *cs, int c)
Returns pointer to first occurrence of 'c' in 'cs' or NULL.

STRCMP
int strcmp(const char *cs, const char *ct)
Returns zero if cs == ct.

STRCPY
char *strcpy(char *s, const char *ct)
Copies 'ct' to 's', including '\0', returns 's'.

STRCSPN
size_t strcspn(const char *cs, const char *ct)

STRERROR
char *strerror(n)
Returns pointer to string defining error 'n'.

STRLEN
size_t strlen(const char *cs)
Returns length of 'cs'.

STRNCAT
char *strncat(char *s, const char *ct, size_t n)

STRNCMP
int strncmp(const char *cs, const char *ct, size_t n)
Returns zero if 'n' characters of both strings are equal.

STRNCPY
char *strncpy(char *s, const char *ct, size_t n)

STRPBRK
char *strpbrk(const char *cs, const char *ct)
Return pointer to first occurrence of any character in 'ct' or NULL.

STRRCHR
char *strrchr(const char *cs, int c)
Returns pointer to last occurrence of 'c' in 'cs' or NULL.

STRSPN
size_t strspn(const char *cs, const char *ct)

STRSTR
char *strstr(const char *cs, const char *ct)
Returns pointer to first occurrence of string 'ct' in 'cs' or NULL.

STRTOK
char *strtok(char *s, const char *ct)
Searches 's' for tokens delimited by characters from 'ct'. Returns NULL when no further tokens found.

MATH FUNCTIONS <math.h>

In the following list, 'x' and 'y' are type 'double', n is type 'int'. All angles for trigonometric functions are in radians.

sin(x)
cos(x)
tan(x)
asin(x)
acos(x)
atan(x)
atan2(y,x)
sinh(x)
cosh(x)
tanh(x)
exp(x)

```
log(x)
log10(x)
pow(x,y)
sqrt(x)
ceil(x)
floor(x)
fabs(x)
ldexp(x,n)
frexp(x, int *exp)
modf(x, double *ip)
fmod(x,y)
```

CHARACTER TESTS <ctype.h>

All functions in <ctype.h> take a character and return an 'int' which represents a true (non-zero) or false (zero) result.

```
isalnum(int c)     alpha-numeric character
isalpha(int c)     upper or lower-case letter
iscntrl(int c)     control character
isdigit(int c)     decimal digit
isgraph(int c)     !space
islower(int c)     lower-case
isprint(int c)     printing character, including space
ispunct(int c)     !(space || letter || digit)
isspace(int c)     space, tab, newline
isupper(int c)     upper-case
isxdigit(int c)    hexadecimal digit
```

DATA AND TIME FUNCTIONS <time.h>

Many functions in <time.h> utilize a data structure which has the following format:

```
struct tm

   int tm_sec;
   int tm_min;
   int tm_hour;
   int tm_mday;
   int tm_mon;
   int tm_year;
   int tm_wday;
   int tm_yday;
   int tm_isdst;   // Daylight Savings Time flag
;
```

ASCTIME
char *asctime(const struct tm *tp)
Converst time into standard string; "Fri May 26 19:27:30 1995\n".

CLOCK
clock_t clock(void)
Returns processor time used by the program.

CTIME
char *ctime(const time_t *tp)
Converts calendar time to local time.

DIFFTIME
double difftime(time_t time2, time_t time1)
Returns difference in two times in seconds.

GMTIME
struct tm *gmtime(const time_t *tp)
Converts calendar time into Coordinated Universal Time (UTC).

LOCALTIME
struct tm *localtime(const time_t *tp)
Converts calendar time into local time.

MKTIME
time_t mktime(struct tm *tp)
Converts local time to calendar time.

STRFTIME
size_t strftime(char *s,size_t smax,const char *fmt,const struct tm *tp)
Formats date and time data from *tp into string 's'. Available formats include:

- %a abbriviated weekday name
- %A full weekday
- %b abbreviated month
- %B full month
- %c date and time
- %d day of month (01-31)
- %H hour (24 hr)
- %I hour (12 hr)
- %j day of year
- %m month
- %M minute
- %p AM or PM
- %S second
- %U week number (Sunday 1st day of week)
- %w weekday (0-6)
- %W week number (Monday 1st day of week)
- %x date
- %X time
- %y year without century
- %Y century
- %Z time zone name
- %% %

TIME
time_t time(time_t *tp)
Returns current calendar time. If 'tp' is not NULL, return value is also assigned to 'tp'.

VARIABLE ARGUMENT LISTS <stdarg.h>

Use the functions in <stdarg.h> to retrieve arguments from a variable length function argument list. An example of using variable argument lists can be found in Section 'Functions'. Before using functions, you must declare a variable of type 'va_list' as follows:

```
va_list ap;
```

You then use the 'va_start' macro to initialize. Afterwords, each subsequent call to 'va_arg' returns the next argument. Finally, call 'va_end'.

VA_START

void va_start(va_list ap, lastarg)
Provide the last argument in a variable length function list as 'lastarg' to initialize va_list variable.

VA_ARG
type va_arg(va_list ap, type)
Returns next argument in variable length function list. You must supply the data type in 'type'. va_arg returns type specified in 'type'.

VA_END
void va_end(va_list ap)
Execute after arguments have been processed, but prior to exiting function.

IMPLEMENTATION SPECIFICS <limits.h> and <float.h>

The following constants are defined in <limits.h>:

- CHAR_BIT
- CHAR_MAX
- CHAR_MIN
- INT_MAX
- INT_MIN
- LONG_MAX
- LONG_MIN
- SCHAR_MAX
- SCHAR_MIN
- SHRT_MAX
- SHRT_MIN
- UCHAR_MAX
- UINT_MAX
- ULONG_MAX
- USHRT_MAX

The following constants are defined in <float.h>:

- FLT_RADIX
- FLT_ROUNDS
- FLT_DIG
- FLT_EPSILON
- FLT_MANT_DIG
- FLT_MAX
- FLT_MAX_EXP
- FLT_MIN
- FLT_MIN_EXP
- DBL_DIG
- DBL_EPSILON
- DBL_MANT_DIG
- DBL_MAX
- DBL_MAX_EXP
- DBL_MIN
- DBL_MIN_EXP

DIAGNOSTICS <assert.h>

Built-in diagnostic macro which is rarely used but included just to be complete.

ASSERT

void assert(int expr)
If 'expr' is zero, prints error message on stderr. If NDEBUG is defined, assert macro is ignored.

NON-LOCAL JUMPS <setjmp.h>

Bad idea which is rarely used but included just to be complete.

SETJMP
int setjmp(jmp_buf env)
Saves state information for later use by 'longjmp'.

LONGJMP
void longjmp(jmp_buf env, int val)
Restores state saved by most recent call to setjmp. Execution resumes as if the setjmp function had just executed and returned the non-zero value val. Function containing setjmp must not have terminated.

SIGNALS <signal.h>

Precursor to C++ exception handling? Rarely used but included just to be complete.

SIGNAL
void (*signal(int sig, void (*handler)(int)))(int)
Calls function pointed to by handler with argument of the type of signal.
- SIG_DFL default behavior: handler ignored
- SIG_IGN ignore signal: handler ignored
- SIGABRT abnormal termination
- SIGFPE arithmetic error
- SIGILL illegal instruction
- SIGINT interrupt
- SIGSEGV illegal storage access
- SIGTERM termination request

RAISE
int raise(int sig)
Sends the signal 'sig' to the program, returns non-zero if unsuccessful.