

C0 Libraries

15-122: Principles of Imperative Computation
Frank Pfenning

September 2011
(Updated Tue Dec 11 2011; [updates](#))

1 Introduction

This document describes the standard libraries available for use in the C0 language implementation. All libraries can be used with the cc0 compiler, and most can also be used with coin. Please consult the [C0 Tutorial](#) and [C0 Language Reference](#) for more information on C0.

Libraries can be include on the command line using `-l lib` or in files with `#use <lib>`. The latter is recommended to make source files less dependent on context. Each library will be loaded at most once.

2 Input/Output

2.1 conio

The conio library contains functions for performing basic console input and output. Note that output may be *buffered*, which means output does not necessarily appear on the console until a newline character is printed or `flush()` is called.

```
void print(string s);      /* print s to standard output */
void println(string s);    /* print s with trailing newline */
void printint(int i);      /* print i to standard output */
void printbool(bool b);   /* print b to standard output */
void printchar(char c);   /* print c to standard output */
void flush();             /* flush standard output */

bool eof();               /* test end-of-file on standard input */
```

```
string readline()          /* read a line from standard input */
/*@requires !eof();        /* do not include the trailing newline */
;
```

2.2 file

The file library contains functions for reading files, line by line. File handles are represented with the `file_t` type. The handle contains an internal position which ranges from 0 to the logical size of the file in bytes. File handles should be closed explicitly when they are no longer needed to release system resources.

```
struct file;
typedef struct file* file_t; /* file handle or NULL */

/* Create a handle for reading from the file given by the specified
 * path, NULL if the file cannot be opened for reading.
 */
file_t file_read(string path);

/* Release any resources associated with the file handle.
 * This function should not be invoked twice on the same handle.
 */
void file_close(file_t f)
    /*@requires f != NULL;
    ;

/* Test if we have read the whole file. */
bool file_eof(file_t f)
    /*@requires f != NULL;
    ;

/* Read a line from the given file (without the trailing newline)
 * and advance the handle's internal position by one line. The
 * contract requires that the handle is not at the end-of-file,
 * so this must be checked before (with file_eof).
 */
string file_readline(file_t f)
    /*@requires f != NULL;
    /*@requires !file_eof(f);
    ;
```

2.3 args

The args library provides function for basic parsing of command line arguments provided to the executable provided by the cc0 compiler. The args library is not supported by coin since it never produces an executable.

```
/* Add a flag with the given name. During parsing if that flag is set
 * (with -name on the command line), it writes the value true to the
 * location given by ptr.
 */
void args_flag(string name, bool *ptr)
    //@requires ptr != NULL;
    ;

/* Add an integer option with the given name. During parsing if that
 * option is given (with -name <int>) it attempts to parse it as an
 * integer and write it to the location given by ptr.
 */
void args_int(string name, int *ptr)
    //@requires ptr != NULL;
    ;

/* Add an string option with the given name. During parsing if that
 * option is given (with -name <string>) it write it to the location
 * given by ptr.
 */
void args_string(string name, string *ptr)
    //@requires ptr != NULL;
    ;

struct args {
    int argc;
    string[] argv;
};
typedef struct args* args_t;

/* Parse the program's arguments according to any flags and options
 * set up by previous calls. Any unrecognized arguments are
 * returned in order in \result->argv.
 *
 * If there is an error, args_parse will return NULL.
 */
args_t args_parse()
    //@ensures \result == NULL || \result->argc == \length(\result->argv);
    ;
```

3 Strings

3.1 parse

The parse library provides function to convert strings to booleans or integers. This is useful, for example, if you want to convert strings read from a file into integers.

```
/* Attempt to parse "true" or "false" from s and return
 * a pointer to the result or NULL if not of that form
 */
bool* parse_bool(string s);

/* Attempt to parse an integer from the given string.
 * For base > 10, the letters A-Z (case insignificant) are used as digits.
 * Return NULL if s cannot be completely parsed to an int in the given
 * base.
 */
int* parse_int(string s, int base)
    //@requires 2 <= base && base <= 36;
    ;
```

3.2 string

The `string` library contains a few basic functions for working with strings consisting of ASCII characters. For most nontrivial tasks, it is best to convert back and forth between characters arrays and strings, using the `string_to_chararray` and `string_from_chararray` functions. The two are identified in C, but distinguished in C0 to allow a type-safe and memory-safe implementation. Note that character arrays should contain a trailing `'\0'` character not present in the corresponding strings.

```
/* Return length of s, in characters.
 * May be an O(n) operation.
 */
int string_length(string s);

/* Return s[idx] and abort if the idx is out of bound.
 * May be an O(n) operation.
 */
char string_charat(string s, int idx)
    //@requires 0 <= idx && idx < string_length(s);
    ;

/* Return a new string that is the result of
 * concatenating a and b.
 */
string string_join(string a, string b)
    //@ensures string_length(\result) == string_length(a) + string_length(b);
    ;

/* Returns the substring composed of the characters of s beginning at
 * the index given by start and continuing up to but not including
 * the index given by end. If end == start, the empty string is returned.
 * Aborts if start or end are out of bounds, or end < start.
 */
string string_sub(string a, int start, int end)
    //@requires 0 <= start && start <= end && end <= string_length(a);
    //@ensures string_length(\result) == end - start;
    ;

/* Compare strings lexicographically */
bool string_equal(string a, string b);
int string_compare(string a, string b)
    //@ensures -1 <= \result && \result <= 1;
    ;
```

```
/* Create strings from other basic values */
string string_fromint(int i);
string string_frombool(bool b);
string string_fromchar(char c)
    //@requires c != '\0';
    //@ensures string_length(\result) == 1;
;

/* Convert every uppercase character A-Z to lowercase a-z */
string string_tolower(string s);

/* Check if character array is properly \0-terminated */
bool string_terminated(char[] A, int n)
    //@requires 0 <= n && n <= \length(A);
;

/* Construct a '\0'-terminated character array from s */
char[] string_to_chararray(string s)
    //@ensures \length(\result) >= string_length(s) + 1;
    //@ensures string_terminated(\result, string_length(s) + 1);
;

/* Construct a string from the the array A
 * up to (but not including) the terminating '\0'
 */
string string_from_chararray(char[] A)
    //@requires string_terminated(A, \length(A));
    //@ensures string_length(\result) + 1 <= \length(A);
;

/* Convert between characters and their ASCII value */
int char_ord(char c)
    //@ensures 0 <= \result && \result <= 127;
;
char char_chr(int n)
    //@requires 0 <= n && n <= 127;
;
```

4 Images

4.1 img

The `img` library defines a type for two-dimensional images represented with 4-channel color—alpha, red, green and blue—packed into one `int`. It defines an image type `image_t`. Like file handles, images must be explicitly destroyed when they are no longer needed; the garbage collector will not be able to free them.

```
struct image;
typedef struct image* image_t;

/* Retrieves the width of the given image */
int image_width(image_t image)
    //@requires image != NULL;
    //@ensures \result > 0;
    ;

/* Retrieves the height of the given image */
int image_height(image_t image)
    //@requires image != NULL;
    //@ensures \result > 0;
    ;

/* Creates an ARGB image with dimensions width * height */
image_t image_create(int width, int height)
    //@requires 0 < width && 0 < height;
    //@ensures \result != NULL;
    //@ensures image_width(\result) == width && image_height(\result) == height;
    ;

/* Copies an existing image */
image_t image_clone(image_t image)
    //@requires image != NULL;
    //@ensures image_width(\result) == image_width(image);
    //@ensures image_height(\result) == image_height(image);
    ;

/* Deprecated. Functions as a no-op since images are now garbage collected */
void image_destroy(image_t image);

/* Returns a copy of a subrectangle of the given image. The new image has
 * dimensions width * height. If part of the given rectangle is not contained
 * within the given image, those pixels are assumed to be transparent black.
```

```
*/
image_t image_subimage(image_t image, int x, int y, int width, int height)
    //@requires image != NULL;
    //@ensures image_width(\result) == width;
    //@ensures image_height(\result) == height;
    ;

/* Loads an image from the given path and convert it if need be to an ARGB image.
 * If the file does not exist, it returns NULL.  Aborts, if the file has
 * the wrong format.
 */
image_t image_load(string path);

/* Saves the given image to disk, inferring the file type from the file
 * extension given in the path.
 */
void image_save(image_t image, string path)
    //@requires image != NULL;
    ;

/* Returns an array of pixels representing the image. The pixels are given line
 * by line so a pixel at (x,y) would be located at y*image_width(image) + x. Any
 * writes to the array will be reflected in calls to image_save, image_clone and
 * image_subimage. The channels are encoded as 0xAARRGGBB.
 */
int[] image_data(image_t image)
    //@requires image != NULL;
    //@ensures \length(\result) == image_width(image) * image_height(image);
    ;
```

5 C0-implemented libraries

A few libraries are included as part of the default C0 distribution but are implemented in C0, not in C. Both the interface (.h0) and the implementation (.c0) of these libraries are provided in the lib directory.

5.1 rand

```
typedef struct rand* rand_t;
rand_t init_rand (int seed)
    //@requires seed != 0;
    ;
int rand(rand_t gen);
```


5.2 util

```
/* C0 constants */
int int_size()
//@ensures \result == 4;
;
int int_max()
//@ensures \result == 2147483647;
;
int int_min()
//@ensures \result == -2147483648;
;

/* Absolute value */
int abs(int x)
//@requires x > int_min();
//@ensures \result >= 0;
//@ensures \result == (x < 0 ? -x : x);
;

/* Returns the hexadecimal representation of the given integer as a
 * string (without the leading 0x) */
string int2hex(int x);
```

6 Updates

Rev. C0.0011, Jan 06, 2012. Image Library. `image_destroy` has been deprecated, since images are now garbage collected. Contracts on image functions have been sharpened to require strictly positive height and width. `image_load` returns `NULL` if the file does not exist or is not readable.

Rev. C0.0113, Sep 27, 2012. Conio Library. Function `eof` has been added to test end-of-file along standard input. This fixes a problem with `^D` during `readline` and end of stream for shell redirect to `stdin`. `readline` now requires `!eof()`.

Rev. C0.0167, Dec 8, 2012. Conio Library. Function `error` has been removed from the conio library, since it now exists as a language primitive.

Rev. C0.0273, Feb 1, 2013. Add `util` and `rand` libraries.