C_0 Library Documentation

Rob Arnold

September 19, 2010

These are the standard libraries as of this publication.

1 Input/Output

1.1 `conio`

The `conio` library contains functions for performing basic console input and output.

```
void print(string s)
```

Prints `s` to standard output.

```
void println(string s)
```

Same as `print` but unconditionally prints a newline `\n`.

```
void printint(int i)
```

A simple convenience function which converts `i` to a string and calls `print`.

```
string readline()
```

Parses a newline (`\n` or `\r\n`) delimited sequence of characters from standard input and returns them as a string.

1.2 `file`

The `file` library contains functions for reading lines out of files. Files handles are represented by 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 when they are no longer needed. The program must close them explicitly - garbage collection of the file handle will not close it.

```
file_t file_read(string path)
```

Creates a handle for reading from the file given by the specified path. If the file cannot be opened for reading, the program aborts.

```
void file_close(file_t f)
```

Releases any resources associated with the file handle. This function should not be invoked twice on the same handle.

```
bool file_eof(file_t f)
```

Returns true if the internal position of the handle is the size of the file.

```
string file_readline(file_t f)
```

Reads a line from the given file, advancing the handle's internal position by the number of characters in the returned string plus the delimeters. Lines are denoted by the `\n` or `\r\n` characters or the end of the file.

1.3 `args`

The `args` library provides functions for basic argument parsing. There are several functions that set up the description of the argument schema and then a single function (`args_parse`) which performs the parsing.

```
void args_flag(string name, bool *ptr)
```

Describes a simple boolean flag. If present on the command line, `args_parse` sets `*ptr` to `true`.

```
void args_int(string name, int *ptr)
```

Describes a switch expecting an integer of the form accepted by `parse_integer` with `base = 0`. If present on the command line,
`args_parse` sets `*ptr` to the value parsed from the switch. If the value could not be parsed, it is not set.

```
void args_string(string name, string *ptr)
```

Describes a switch expecting some additional argument. If present on the command line, `args_parse` sets `*ptr` to the argument.

```
string[] args_parse()
```

Attempts to parse the command line arguments given to the program by the operating system. Arguments that indicate a switch consume the next argument. Arguments that are not matched to switches or flags are considered positional arguments and are returned in an array.

2 Data manipulation

2.1 `parse`

The `parse` library provides two functions to parse integers and booleans. These functions return pointers to two structs: `struct parsed_bool` and `struct parsed_int`. `struct parsed_bool` is has the following members:

`bool parsed`	Indicates if the string was successfully parsed
`bool value`	If the `parsed` field is true, holds the parsed value

`struct parsed_int` has the same members except that `value` is of type `int`.

```
struct parsed_bool *parse_bool(string s)
```

Attempts to parse `s` into a value of type `bool`. Accepts `"true"` and `"false"` as valid strings.

```
struct parsed_int *parse_int(string s, int b)
```

Attempts to parse `s` as a number written in base `b`. Supported bases include 8, 10 and 16. If `b` is 0, the base of the number is inferred from the leading digits. `0x` indicates that the base is 16, otherwise `0` indicates base 8 and any other digit indicates base 10.

If the number is too large to be represented as an `int`, number is not parsed.

2.2 string

The `string` library contains a few basic routines for working with strings and characters.

```
int string_length(string s)
```

Returns the number of characters in `s`.

```
char string_charat(string s, int n)
```

Returns the `n`th character in `s`. If `n` is less than zero or greater than the length of the string, the program aborts.

```
string string_join(string a, string b)
```

Returns a string containing the contents of `b` appended to the contents of `a`.

```
string string_sub(string s, int start, int end)
```

Returns the substring composed of the characters of `s` beginning at index given by `start` and continuing up to but not including the index given by end If `end <= start`, the empty string is returned If `end < 0` or `end >` the length of the string, it is treated as though it were equal to the length of the string. If `start < 0` the empty string is returned.

```
bool string_equal(string a, string b)
```

Returns `true` if the contents of `a` and `b` are equal and `false` otherwise.

```
int string_compare(string a, string b)
```

Compares `a` and `b` lexicographically. If `a` comes before `b`, then the return value is negative. If `string_equal(a,b)` is `true`, the return value is 0. Otherwise the return value is greater than 0.

```
bool char_equal(char a, char b)
```

Returns whether or not the two characters are identical.

```
int char_compare(char a, char b)
```

Compares the two characters according to their ASCII encoding. If the two characters have the same encoding, the return value is 0. If `char_ord(a) < char_ord(b)`, then the return value is less than 0. Otherwise it is greater than 0.

```
string string_frombool(bool b)
```

Returns a canonical representation of `b` as a string. The returned value will always be parsed by `parse_bool` into a value equal to `b`.

```
string string_fromint(int i)
```

Returns a canonical representation of `i` as a string. The returned value will always be parsed by `parse_int` into a value equal to `i`.

```
string string_tolower(string s)
```

Returns a string containing the same character sequence as `s` but with each upper case character replaced by its lower case version.

```
char[] string_to_chararray(string s)
```

Returns the characters in `s` as an array. The length of the array is one more than the length of `s`. The last character in the array is `'\0'`.

```
string string_from_chararray(char[] A)
```

Returns a string containing the characters from `A` in it. The last character of the array must be equal to `'\0'`. The program will abort if it is not.

```
int char_ord(char c)
```

Returns an integer representing the ASCII encoding of `c`.

```
char char_chr(int n)
```

Decodes `n` as an ASCII character and returns the result. If `n` cannot be decoded as valid ASCII, the program aborts.

3 Images

3.1 `img`

The `img` library defines a type for two dimensional images represented as pixels with 4 color channels - alpha, red, green and blue - packed into one `int`. It defines an image type `image_t`. Images must be explicitly destroyed when they are no longer needed with the `image_destroy` function.

```
image_t image_create(int width, int height)
```

Creates an image with the given width and height. The default pixel color is transparent black. `width` and `height` must be positive.

```
image_t image_clone(image_t image)
```

Creates a copy of the image.

```
void image_destroy(image_t image)
```

Releases any internal resources associated with `image`. The array returned by a previous `image_data` call will remain valid however any subsequent calls using `image` will cause the program to abort.

```
image_t image_subimage(image_t image, int x, int y, int w, int h)
```

Creates a partial copy of `image` using the rectangle as the source coordinates in `image`. Any parts of the given rectangle that are not contained in `image` are treated as transparent black.

```
image_t image_load(string path)
```

Loads an image from the file given by `path` and converts it if need be to an ARGB image. If the file cannot be found, the program aborts.

```
void image_save(image_t image, string path)
```

Saves `image` to the file given by `path`. If the file cannot be written, the program aborts.

```
int image_width(image_t image)
```

Returns the width in pixels of `image`.

```
int image_height(image_t image)
```

Returns the height in pixels of `image`.

```
int[] image_data(image_t image)
```

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`.