Sudoku

1.0

Generated by Doxygen 1.8.20

Chapter 1

Contributors

Original Author:

- Dave Gamble

Current Maintainer:

- Max Bruckner
- Alan Wang

Contributors:

- Ajay Bhargav
- Alper Akcan
- Andrew Tang
- Anton Sergeev
- Benbuck Nason
- Bernt Johan Damslora
- Bob Kocisko
- Christian Schulze
- Casperinous
- ChenYuan
- Debora Grosse
- dieyushi
- Dngwén Huáng ()
- Donough Liu
- Erez Oxman

- Eswar Yaganti
- Evan Todd
- Fabrice Fontaine
- Ian Mobley
- Irwan Djadjadi
- IvanVoid
- Jakub Wilk
- Jiri Zouhar
- Jonathan Fether
- Julian Ste
- Julián Vásquez
- Kevin Branigan
- Kyle Chisholm
- Linus Wallgren
- Mateusz Szafoni
- Mike Pontillo
- Mike Jerris
- Mike Robinson
- myd7349
- NancyLi1013
- Paulo Antonio Alvarez
- Pawe Malowany
- Pawel Winogrodzki
- prefetchnta
- Rafael Leal Dias
- Randy
- raiden00pl
- Robin Mallinson
- Rod Vagg
- Roland Meertens
- Romain Porte
- SANJEEV BA
- Sang-Heon Jeon
- Simon Sobisch
- Simon Ricaldone
- Square789

- Stephan Gatzka
- Vemake
- Wei Tan
- Weston Schmidt
- xiaomianhehe
- yangfl
- yuta-oxo
- Zach Hindes
- Zhao Zhixu

And probably more people on SourceForge

Also thanks to all the people who reported bugs and suggested new features.

Chapter 2

cJSON

Ultralightweight JSON parser in ANSI C.

2.1 Table of contents

2.2 License

MIT License

2.3 Usage

2.3.1 Welcome to cJSON.

cJSON aims to be the dumbest possible parser that you can get your job done with. It's a single file of C, and a single header file.

JSON is described best here: `http://www.json.org/` It's like XML, but fat-free. You use it to move data around, store things, or just generally represent your program's state.

As a library, cJSON exists to take away as much legwork as it can, but not get in your way. As a point of pragmatism (i.e. ignoring the truth), I'm going to say that you can use it in one of two modes: Auto and Manual. Let's have a quick run-through.

I lifted some JSON from this page: `http://www.json.org/fatfree.html` That page inspired me to write cJSON, which is a parser that tries to share the same philosophy as JSON itself. Simple, dumb, out of the way.

2.3.2 Building

There are several ways to incorporate cJSON into your project.

2.3.2.1 copying the source

Because the entire library is only one C file and one header file, you can just copy `cJSON.h` and `cJSON.c` to your projects source and start using it.

cJSON is written in ANSI C (C89) in order to support as many platforms and compilers as possible.

2.3.2.2 CMake

With CMake, cJSON supports a full blown build system. This way you get the most features. CMake with an equal or higher version than 2.8.5 is supported. With CMake it is recommended to do an out of tree build, meaning the compiled files are put in a directory separate from the source files. So in order to build cJSON with CMake on a Unix platform, make a `build` directory and run CMake inside it.

```
mkdir build
cd build
cmake ..
```

This will create a Makefile and a bunch of other files. You can then compile it:

```
make
```

And install it with `make install` if you want. By default it installs the headers `/usr/local/include/cjson` and the libraries to `/usr/local/lib`. It also installs files for pkg-config to make it easier to detect and use an existing installation of CMake. And it installs CMake config files, that can be used by other CMake based projects to discover the library.

You can change the build process with a list of different options that you can pass to CMake. Turn them on with `On` and off with `Off`:

- `-DENABLE_CJSON_TEST=On`: Enable building the tests. (on by default)
- `-DENABLE_CJSON_UTILS=On`: Enable building cJSON_Utils. (off by default)
- `-DENABLE_TARGET_EXPORT=On`: Enable the export of CMake targets. Turn off if it makes problems. (on by default)
- `-DENABLE_CUSTOM_COMPILER_FLAGS=On`: Enable custom compiler flags (currently for Clang, GCC and MSVC). Turn off if it makes problems. (on by default)
- `-DENABLE_VALGRIND=On`: Run tests with `valgrind`. (off by default)
- `-DENABLE_SANITIZERS=On`: Compile cJSON with `AddressSanitizer` and `Undefined↩ BehaviorSanitizer` enabled (if possible). (off by default)
- `-DENABLE_SAFE_STACK`: Enable the `SafeStack` instrumentation pass. Currently only works with the Clang compiler. (off by default)
- `-DBUILD_SHARED_LIBS=On`: Build the shared libraries. (on by default)
- `-DBUILD_SHARED_AND_STATIC_LIBS=On`: Build both shared and static libraries. (off by default)
- `-DCMAKE_INSTALL_PREFIX=/usr`: Set a prefix for the installation.
- `-DENABLE_LOCALES=On`: Enable the usage of localeconv method. (on by default)
- `-DCJSON_OVERRIDE_BUILD_SHARED_LIBS=On`: Enable overriding the value of `BUILD_SHARED↩ _LIBS` with `-DCJSON_BUILD_SHARED_LIBS`.

If you are packaging cJSON for a distribution of Linux, you would probably take these steps for example:

```
mkdir build
cd build
cmake .. -DENABLE_CJSON_UTILS=On -DENABLE_CJSON_TEST=Off -DCMAKE_INSTALL_PREFIX=/usr
make
make DESTDIR=$pkgdir install
```

On Windows CMake is usually used to create a Visual Studio solution file by running it inside the Developer Command Prompt for Visual Studio, for exact steps follow the official documentation from CMake and Microsoft and use the online search engine of your choice. The descriptions of the the options above still generally apply, although not all of them work on Windows.

2.3.2.3 Makefile

NOTE: This Method is deprecated. Use CMake if at all possible. Makefile support is limited to fixing bugs.

If you don't have CMake available, but still have GNU make. You can use the makefile to build cJSON:

Run this command in the directory with the source code and it will automatically compile static and shared libraries and a little test program (not the full test suite).

```
make all
```

If you want, you can install the compiled library to your system using `make install`. By default it will install the headers in `/usr/local/include/cjson` and the libraries in `/usr/local/lib`. But you can change this behavior by setting the `PREFIX` and `DESTDIR` variables: `make PREFIX=/usr DESTDIR=temp install`. And uninstall them with: `make PREFIX=/usr DESTDIR=temp uninstall`.

2.3.2.4 Vcpkg

You can download and install cJSON using the `vcpkg` dependency manager:

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
./bootstrap-vcpkg.sh
./vcpkg integrate install
vcpkg install cjson
```

The cJSON port in vcpkg is kept up to date by Microsoft team members and community contributors. If the version is out of date, please `create an issue or pull request` on the vcpkg repository.

2.3.3 Including cJSON

If you installed it via CMake or the Makefile, you can include cJSON like this:

```
#include <cjson/cJSON.h>
```

2.3.4 Data Structure

cJSON represents JSON data using the `cJSON` struct data type:

```
/* The cJSON structure: */
typedef struct cJSON
{
    struct cJSON *next;
    struct cJSON *prev;
    struct cJSON *child;
    int type;
    char *valuestring;
    /* writing to valueint is DEPRECATED, use cJSON_SetNumberValue instead */
    int valueint;
    double valuedouble;
    char *string;
} cJSON;
```

An item of this type represents a JSON value. The type is stored in `type` as a bit-flag (**this means that you cannot find out the type by just comparing the value of `type`**).

To check the type of an item, use the corresponding `cJSON_Is...` function. It does a `NULL` check followed by a type check and returns a boolean value if the item is of this type.

The type can be one of the following:

- `cJSON_Invalid` (check with `cJSON_IsInvalid`): Represents an invalid item that doesn't contain any value. You automatically have this type if you set the item to all zero bytes.
- `cJSON_False` (check with `cJSON_IsFalse`): Represents a `false` boolean value. You can also check for boolean values in general with `cJSON_IsBool`.
- `cJSON_True` (check with `cJSON_IsTrue`): Represents a `true` boolean value. You can also check for boolean values in general with `cJSON_IsBool`.
- `cJSON_NULL` (check with `cJSON_IsNull`): Represents a `null` value.
- `cJSON_Number` (check with `cJSON_IsNumber`): Represents a number value. The value is stored as a double in `valuedouble` and also in `valueint`. If the number is outside of the range of an integer, `INT_MAX` or `INT_MIN` are used for `valueint`.
- `cJSON_String` (check with `cJSON_IsString`): Represents a string value. It is stored in the form of a zero terminated string in `valuestring`.
- `cJSON_Array` (check with `cJSON_IsArray`): Represent an array value. This is implemented by pointing `child` to a linked list of `cJSON` items that represent the values in the array. The elements are linked together using `next` and `prev`, where the first element has `prev.next == NULL` and the last element `next == NULL`.
- `cJSON_Object` (check with `cJSON_IsObject`): Represents an object value. Objects are stored same way as an array, the only difference is that the items in the object store their keys in `string`.
- `cJSON_Raw` (check with `cJSON_IsRaw`): Represents any kind of JSON that is stored as a zero terminated array of characters in `valuestring`. This can be used, for example, to avoid printing the same static JSON over and over again to save performance. cJSON will never create this type when parsing. Also note that cJSON doesn't check if it is valid JSON.

Additionally there are the following two flags:

- `cJSON_IsReference`: Specifies that the item that `child` points to and/or `valuestring` is not owned by this item, it is only a reference. So `cJSON_Delete` and other functions will only deallocate this item, not its `child`/`valuestring`.
- `cJSON_StringIsConst`: This means that `string` points to a constant string. This means that `cJS↩ ON_Delete` and other functions will not try to deallocate `string`.

2.3.5 Working with the data structure

For every value type there is a `cJSON_Create...` function that can be used to create an item of that type. All of these will allocate a `cJSON` struct that can later be deleted with `cJSON_Delete`. Note that you have to delete them at some point, otherwise you will get a memory leak.
Important: If you have added an item to an array or an object already, you **mustn't** delete it with `cJSON_Delete`. Adding it to an array or object transfers its ownership so that when that array or object is deleted, it gets deleted as well. You also could use `cJSON_SetValuestring` to change a `cJSON_String`'s `valuestring`, and you needn't to free the previous `valuestring` manually.

2.3.5.1 Basic types

- **null** is created with `cJSON_CreateNull`
- **booleans** are created with `cJSON_CreateTrue`, `cJSON_CreateFalse` or `cJSON_CreateBool`
- **numbers** are created with `cJSON_CreateNumber`. This will set both `valuedouble` and `valueint`. If the number is outside of the range of an integer, `INT_MAX` or `INT_MIN` are used for `valueint`
- **strings** are created with `cJSON_CreateString` (copies the string) or with `cJSON_CreateString↩ Reference` (directly points to the string. This means that `valuestring` won't be deleted by `cJSON_↩ Delete` and you are responsible for its lifetime, useful for constants)

2.3.5.2 Arrays

You can create an empty array with `cJSON_CreateArray`. `cJSON_CreateArrayReference` can be used to create an array that doesn't "own" its content, so its content doesn't get deleted by `cJSON_Delete`.

To add items to an array, use `cJSON_AddItemToArray` to append items to the end. Using `cJSON_AddItemReferenceToArray` an element can be added as a reference to another item, array or string. This means that `cJSON_Delete` will not delete that items `child` or `valuestring` properties, so no double frees are occurring if they are already used elsewhere. To insert items in the middle, use `cJSON_InsertItemInArray`. It will insert an item at the given 0 based index and shift all the existing items to the right.

If you want to take an item out of an array at a given index and continue using it, use `cJSON_DetachItemFromArray`, it will return the detached item, so be sure to assign it to a pointer, otherwise you will have a memory leak.

Deleting items is done with `cJSON_DeleteItemFromArray`. It works like `cJSON_DetachItemFromArray`, but deletes the detached item via `cJSON_Delete`.

You can also replace an item in an array in place. Either with `cJSON_ReplaceItemInArray` using an index or with `cJSON_ReplaceItemViaPointer` given a pointer to an element. `cJSON_ReplaceItemViaPointer` will return `0` if it fails. What this does internally is to detach the old item, delete it and insert the new item in its place.

To get the size of an array, use `cJSON_GetArraySize`. Use `cJSON_GetArrayItem` to get an element at a given index.

Because an array is stored as a linked list, iterating it via index is inefficient ($O(n^2)$), so you can iterate over an array using the `cJSON_ArrayForEach` macro in $O(n)$ time complexity.

2.3.5.3 Objects

You can create an empty object with `cJSON_CreateObject`. `cJSON_CreateObjectReference` can be used to create an object that doesn't "own" its content, so its content doesn't get deleted by `cJSON_Delete`.

To add items to an object, use `cJSON_AddItemToObject`. Use `cJSON_AddItemToObjectCS` to add an item to an object with a name that is a constant or reference (key of the item, `string` in the `cJSON` struct), so that it doesn't get freed by `cJSON_Delete`. Using `cJSON_AddItemReferenceToArray` an element can be added as a reference to another object, array or string. This means that `cJSON_Delete` will not delete that items `child` or `valuestring` properties, so no double frees are occurring if they are already used elsewhere.

If you want to take an item out of an object, use `cJSON_DetachItemFromObjectCaseSensitive`, it will return the detached item, so be sure to assign it to a pointer, otherwise you will have a memory leak.

Deleting items is done with `cJSON_DeleteItemFromObjectCaseSensitive`. It works like `cJSON_DetachItemFromObjectCaseSensitive` followed by `cJSON_Delete`.

You can also replace an item in an object in place. Either with `cJSON_ReplaceItemInObjectCaseSensitive` using a key or with `cJSON_ReplaceItemViaPointer` given a pointer to an element. `cJSON_ReplaceItemViaPointer` will return `0` if it fails. What this does internally is to detach the old item, delete it and insert the new item in its place.

To get the size of an object, you can use `cJSON_GetArraySize`, this works because internally objects are stored as arrays.

If you want to access an item in an object, use `cJSON_GetObjectItemCaseSensitive`.

To iterate over an object, you can use the `cJSON_ArrayForEach` macro the same way as for arrays.

cJSON also provides convenient helper functions for quickly creating a new item and adding it to an object, like `cJSON_AddNullToObject`. They return a pointer to the new item or `NULL` if they failed.

2.3.6 Parsing JSON

Given some JSON in a zero terminated string, you can parse it with `cJSON_Parse`.

```
cJSON *json = cJSON_Parse(string);
```

Given some JSON in a string (whether zero terminated or not), you can parse it with `cJSON_ParseWith←Length`.

```
cJSON *json = cJSON_ParseWithLength(string, buffer_length);
```

It will parse the JSON and allocate a tree of `cJSON` items that represents it. Once it returns, you are fully responsible for deallocating it after use with `cJSON_Delete`.

The allocator used by `cJSON_Parse` is `malloc` and `free` by default but can be changed (globally) with `cJS←ON_InitHooks`.

If an error occurs a pointer to the position of the error in the input string can be accessed using `cJSON_Get←ErrorPtr`. Note though that this can produce race conditions in multithreading scenarios, in that case it is better to use `cJSON_ParseWithOpts` with `return_parse_end`. By default, characters in the input string that follow the parsed JSON will not be considered as an error.

If you want more options, use `cJSON_ParseWithOpts(const char *value, const char **return_parse_end, cJSON_bool require_null_terminated)`. `return_parse_end` returns a pointer to the end of the JSON in the input string or the position that an error occurs at (thereby replacing `cJSON_GetErrorPtr` in a thread safe way). `require_null_terminated`, if set to `1` will make it an error if the input string contains data after the JSON.

If you want more options giving buffer length, use `cJSON_ParseWithLengthOpts(const char *value, size_t buffer_length, const char **return_parse_end, cJSON_bool require←_null_terminated)`.

2.3.7 Printing JSON

Given a tree of `cJSON` items, you can print them as a string using `cJSON_Print`.

```
char *string = cJSON_Print(json);
```

It will allocate a string and print a JSON representation of the tree into it. Once it returns, you are fully responsible for deallocating it after use with your allocator. (usually `free`, depends on what has been set with `cJSON_Init←Hooks`).

`cJSON_Print` will print with whitespace for formatting. If you want to print without formatting, use `cJSON_←PrintUnformatted`.

If you have a rough idea of how big your resulting string will be, you can use `cJSON_PrintBuffered(const cJSON *item, int prebuffer, cJSON_bool fmt)`. `fmt` is a boolean to turn formatting with whitespace on and off. `prebuffer` specifies the first buffer size to use for printing. `cJSON_Print` currently uses 256 bytes for its first buffer size. Once printing runs out of space, a new buffer is allocated and the old gets copied over before printing is continued.

These dynamic buffer allocations can be completely avoided by using `cJSON_PrintPreallocated(cJSON *item, char *buffer, const int length, const cJSON_bool format)`. It takes a buffer to a pointer to print to and its length. If the length is reached, printing will fail and it returns `0`. In case of success, `1` is returned. Note that you should provide 5 bytes more than is actually needed, because cJSON is not 100% accurate in estimating if the provided memory is enough.

2.3.8 Example

In this example we want to build and parse the following JSON:

```
{
    "name": "Awesome 4K",
    "resolutions": [
        {
            "width": 1280,
            "height": 720
        },
        {
            "width": 1920,
            "height": 1080
        },
        {
            "width": 3840,
            "height": 2160
        }
    ]
}
```

2.3.8.1 Printing

Let's build the above JSON and print it to a string:

```
//create a monitor with a list of supported resolutions
//NOTE: Returns a heap allocated string, you are required to free it after use.
char *create_monitor(void)
{
    const unsigned int resolution_numbers[3][2] = {
        {1280, 720},
        {1920, 1080},
        {3840, 2160}
    };
    char *string = NULL;
    cJSON *name = NULL;
    cJSON *resolutions = NULL;
    cJSON *resolution = NULL;
    cJSON *width = NULL;
    cJSON *height = NULL;
    size_t index = 0;
    cJSON *monitor = cJSON_CreateObject();
    if (monitor == NULL)
    {
        goto end;
    }
    name = cJSON_CreateString("Awesome 4K");
    if (name == NULL)
    {
        goto end;
    }
    /* after creation was successful, immediately add it to the monitor,
     * thereby transferring ownership of the pointer to it */
    cJSON_AddItemToObject(monitor, "name", name);
    resolutions = cJSON_CreateArray();
    if (resolutions == NULL)
    {
        goto end;
    }
    cJSON_AddItemToObject(monitor, "resolutions", resolutions);
    for (index = 0; index < (sizeof(resolution_numbers) / (2 * sizeof(int))); ++index)
    {
        resolution = cJSON_CreateObject();
        if (resolution == NULL)
        {
            goto end;
        }
        cJSON_AddItemToArray(resolutions, resolution);
        width = cJSON_CreateNumber(resolution_numbers[index][0]);
        if (width == NULL)
        {
            goto end;
        }
        cJSON_AddItemToObject(resolution, "width", width);
        height = cJSON_CreateNumber(resolution_numbers[index][1]);
        if (height == NULL)
        {
            goto end;
        }
        cJSON_AddItemToObject(resolution, "height", height);
    }
    string = cJSON_Print(monitor);
    if (string == NULL)
    {
        fprintf(stderr, "Failed to print monitor.\n");
    }
end:
    cJSON_Delete(monitor);
    return string;
}
```

Alternatively we can use the `cJSON_Add...ToObject` helper functions to make our lifes a little easier:

```
//NOTE: Returns a heap allocated string, you are required to free it after use.
char *create_monitor_with_helpers(void)
{
    const unsigned int resolution_numbers[3][2] = {
        {1280, 720},
        {1920, 1080},
        {3840, 2160}
    };
    char *string = NULL;
    cJSON *resolutions = NULL;
    size_t index = 0;
    cJSON *monitor = cJSON_CreateObject();
    if (cJSON_AddStringToObject(monitor, "name", "Awesome 4K") == NULL)
    {
        goto end;
    }
    resolutions = cJSON_AddArrayToObject(monitor, "resolutions");
    if (resolutions == NULL)
    {
        goto end;
    }
    for (index = 0; index < (sizeof(resolution_numbers) / (2 * sizeof(int))); ++index)
    {
        cJSON *resolution = cJSON_CreateObject();
        if (cJSON_AddNumberToObject(resolution, "width", resolution_numbers[index][0]) == NULL)
        {
            goto end;
        }
        if (cJSON_AddNumberToObject(resolution, "height", resolution_numbers[index][1]) == NULL)
        {
            goto end;
        }
        cJSON_AddItemToArray(resolutions, resolution);
    }
    string = cJSON_Print(monitor);
    if (string == NULL)
    {
        fprintf(stderr, "Failed to print monitor.\n");
    }
end:
    cJSON_Delete(monitor);
    return string;
}
```

2.3.8.2 Parsing

In this example we will parse a JSON in the above format and check if the monitor supports a Full HD resolution while printing some diagnostic output:

```
/* return 1 if the monitor supports full hd, 0 otherwise */
int supports_full_hd(const char * const monitor)
{
    const cJSON *resolution = NULL;
    const cJSON *resolutions = NULL;
    const cJSON *name = NULL;
    int status = 0;
    cJSON *monitor_json = cJSON_Parse(monitor);
    if (monitor_json == NULL)
    {
        const char *error_ptr = cJSON_GetErrorPtr();
        if (error_ptr != NULL)
        {
            fprintf(stderr, "Error before: %s\n", error_ptr);
        }
        status = 0;
        goto end;
    }
    name = cJSON_GetObjectItemCaseSensitive(monitor_json, "name");
    if (cJSON_IsString(name) && (name->valuestring != NULL))
    {
        printf("Checking monitor \"%s\"\n", name->valuestring);
    }
    resolutions = cJSON_GetObjectItemCaseSensitive(monitor_json, "resolutions");
    cJSON_ArrayForEach(resolution, resolutions)
    {
        cJSON *width = cJSON_GetObjectItemCaseSensitive(resolution, "width");
        cJSON *height = cJSON_GetObjectItemCaseSensitive(resolution, "height");
        if (!cJSON_IsNumber(width) || !cJSON_IsNumber(height))
        {
            status = 0;
            goto end;
        }
        if ((width->valuedouble == 1920) && (height->valuedouble == 1080))
        {
            status = 1;
            goto end;
        }
    }
end:
    cJSON_Delete(monitor_json);
    return status;
}
```

Note that there are no NULL checks except for the result of `cJSON_Parse` because `cJSON_GetObject↩ ItemCaseSensitive` checks for `NULL` inputs already, so a `NULL` value is just propagated and `cJSON_Is↩ Number` and `cJSON_IsString` return `0` if the input is `NULL`.

2.3.9 Caveats

2.3.9.1 Zero Character

cJSON doesn't support strings that contain the zero character '\0'`or`\u0000`. This is impossible with the current API because strings are zero terminated.

2.3.9.2 Character Encoding

cJSON only supports UTF-8 encoded input. In most cases it doesn't reject invalid UTF-8 as input though, it just propagates it through as is. As long as the input doesn't contain invalid UTF-8, the output will always be valid UTF-8.

2.3.9.3 C Standard

cJSON is written in ANSI C (or C89, C90). If your compiler or C library doesn't follow this standard, correct behavior is not guaranteed.

NOTE: ANSI C is not C++ therefore it shouldn't be compiled with a C++ compiler. You can compile it with a C compiler and link it with your C++ code however. Although compiling with a C++ compiler might work, correct behavior is not guaranteed.

2.3.9.4 Floating Point Numbers

cJSON does not officially support any `double` implementations other than IEEE754 double precision floating point numbers. It might still work with other implementations but bugs with these will be considered invalid.

The maximum length of a floating point literal that cJSON supports is currently 63 characters.

2.3.9.5 Deep Nesting Of Arrays And Objects

cJSON doesn't support arrays and objects that are nested too deeply because this would result in a stack overflow. To prevent this cJSON limits the depth to `CJSON_NESTING_LIMIT` which is 1000 by default but can be changed at compile time.

2.3.9.6 Thread Safety

In general cJSON is **not thread safe**.

However it is thread safe under the following conditions:

- `cJSON_GetErrorPtr` is never used (the `return_parse_end` parameter of `cJSON_ParseWithOpts` can be used instead)
- `cJSON_InitHooks` is only ever called before using cJSON in any threads.
- `setlocale` is never called before all calls to cJSON functions have returned.

2.3.9.7 Case Sensitivity

When cJSON was originally created, it didn't follow the JSON standard and didn't make a distinction between uppercase and lowercase letters. If you want the correct, standard compliant, behavior, you need to use the `CaseSensitive` functions where available.

2.3.9.8 Duplicate Object Members

cJSON supports parsing and printing JSON that contains objects that have multiple members with the same name. `cJSON_GetObjectItemCaseSensitive` however will always only return the first one.

2.4 Enjoy cJSON!

- Dave Gamble (original author)
- Max Bruckner and Alan Wang (current maintainer)
- and the other cJSON contributors

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 arg Struct Reference

Data Fields

- struct arg ∗ **next**
- struct arg ∗ **prev**
- int **type**
- int **argv_index**
- int **enabled**

5.1.1 Detailed Description

Definition at line 20 of file generic.h.

The documentation for this struct was generated from the following file:

- src/generic.h

5.2 cJSON Struct Reference

Data Fields

- struct cJSON ∗ **next**
- struct cJSON ∗ **prev**
- struct cJSON ∗ **child**
- int **type**
- char ∗ **valuestring**
- int **valueint**
- double **valuedouble**
- char ∗ **string**

5.2.1 Detailed Description

Definition at line 103 of file cJSON.h.

The documentation for this struct was generated from the following file:

- src/cJSON/cJSON.h

5.3 cJSON_Hooks Struct Reference

Public Member Functions

- void ∗CJSON_CDECL ∗ **malloc_fn** (size_t sz)
- **void** (CJSON_CDECL ∗free_fn)(void ∗ptr)

5.3.1 Detailed Description

Definition at line 125 of file cJSON.h.

The documentation for this struct was generated from the following file:

- src/cJSON/cJSON.h

5.4 error Struct Reference

Data Fields

- const unsigned char ∗ **json**
- size_t **position**

5.4.1 Detailed Description

Definition at line 84 of file cJSON.c.

The documentation for this struct was generated from the following file:

- src/cJSON/cJSON.c

5.5 internal_hooks Struct Reference

Public Member Functions

- void ∗CJSON_CDECL ∗ **allocate** (size_t size)
- **void** (CJSON_CDECL ∗deallocate)(void ∗pointer)
- void ∗CJSON_CDECL ∗ **reallocate** (void ∗pointer, size_t size)

5.5.1 Detailed Description

Definition at line 152 of file cJSON.c.

The documentation for this struct was generated from the following file:

- src/cJSON/cJSON.c

5.6 parse_buffer Struct Reference

Data Fields

- const unsigned char $*$ **content**
- size_t **length**
- size_t **offset**
- size_t **depth**
- internal_hooks **hooks**

5.6.1 Detailed Description

Definition at line 283 of file cJSON.c.

The documentation for this struct was generated from the following file:

- src/cJSON/cJSON.c

5.7 printbuffer Struct Reference

Data Fields

- unsigned char $*$ **buffer**
- size_t **length**
- size_t **offset**
- size_t **depth**
- cJSON_bool **noalloc**
- cJSON_bool **format**
- internal_hooks **hooks**

5.7.1 Detailed Description

Definition at line 423 of file cJSON.c.

The documentation for this struct was generated from the following file:

- src/cJSON/cJSON.c

Chapter 6

File Documentation

6.1 main.c File Reference

```
#include <ncurses.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include "src/generic.h"
#include "src/solving.h"
#include "src/json.h"
#include "src/param.h"
```

Functions

- int **main** (int argc, char ∗argv[ ])

6.2 src/generic.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <ncurses.h>
#include "generic.h"
```

Functions

- void help (void)
- int print_to_file (char ∗str, int ∗array)
- void read_array_from_file (FILE ∗source, int ∗array_before, int ∗array_after)
- int file_not_found (FILE ∗src, char ∗name)
- int is_empty (FILE ∗src)
- long file_size (FILE ∗src)
- void calculate_positions (int ∗x, int ∗y)
- uint32_t create_hash (int ∗array)
- void print_array_ncurses (WINDOW ∗win, int ∗array, int ∗positions_y, int ∗positions_x)
- void print_ncurses (WINDOW ∗win, int ∗array, int ∗positions_y, int ∗positions_x)

Variables

- long **REFRESH_COUNTER** = 0

6.2.1 Function Documentation

6.2.1.1 calculate_positions()

```
void calculate_positions (
             int * x,
             int * y )
```

Generate x and y coordinates for ncurses print

Parameters

x	array for x values
y	array for y values

Definition at line 104 of file generic.c.

6.2.1.2 create_hash()

```
uint32_t create_hash (
             int * array )
```

Create hash from array

Parameters

array	array

Definition at line 121 of file generic.c.

6.2.1.3 file_not_found()

```
int file_not_found (
             FILE * src,
             char * name )
```

Called, when file is not found

Parameters

src	file
name	file name

Definition at line 66 of file generic.c.

6.2.1.4 file_size()

```
long file_size (
             FILE * src )
```

Get file size, then rewind

Parameters

src	file

Definition at line 93 of file generic.c.

6.2.1.5 help()

```
void help (
             void  )
```

Print help

Definition at line 11 of file generic.c.

6.2.1.6 is_empty()

```
int is_empty (
             FILE * src )
```

Check, if file is empty

Parameters

src	file

Definition at line 78 of file generic.c.

6.2.1.7 print_array_ncurses()

```
void print_array_ncurses (
             WINDOW * win,
             int * array,
             int * positions_y,
             int * positions_x )
```

Print array via ncurses liblary

Parameters

win	ncurses window
array	values to print
positions←↩ _y	values for printing in y axis
positions←↩ _x	values for printing in x axis

Definition at line 141 of file generic.c.

6.2.1.8 print_ncurses()

```
void print_ncurses (
             WINDOW * win,
             int * array,
             int * positions_y,
             int * positions_x )
```

Call print_array_nucrses every REFRESH_INTERVAL calls

Parameters

win	ncurses window
array	values to print
positions←↩ _y	values for printing in y axis
positions←↩ _x	values for printing in x axis

Definition at line 159 of file generic.c.

6.2.1.9 print_to_file()

```
int print_to_file (
             char * str,
             int * array )
```

Print array to file

Parameters

str	file name
array	array of values

Definition at line 23 of file generic.c.

6.2.1.10 read_array_from_file()

```
void read_array_from_file (
            FILE * source,
            int * array_before,
            int * array_after )
```

Load array from CSV file

Parameters

source	file
array_before	array before solving
array_after	array after solving

Definition at line 39 of file generic.c.

6.3 src/generic.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <ncurses.h>
```

Data Structures

- struct arg

Macros

- #define **clrscr**() printf("\e[1;1H\e[2J")
- #define **REFRESH_INTERVAL** 1000000

Typedefs

- typedef struct arg **arg_t**

Enumerations

- enum **types** {
  HISTORY, **INPUT**, **OUTPUT**, **FORCE**,
  PREVIOUS }
- enum **enable** { **OFF**, **ON** }

Functions

- void help (void)
- int print_to_file (char ∗, int ∗)
- void read_array_from_file (FILE ∗, int ∗, int ∗)
- int file_not_found (FILE ∗, char ∗)
- int is_empty (FILE ∗)
- long file_size (FILE ∗)
- void calculate_positions (int ∗, int ∗)
- uint32_t create_hash (int ∗)
- void print_array_ncurses (WINDOW ∗, int ∗, int ∗, int ∗)
- void print_ncurses (WINDOW ∗, int ∗, int ∗, int ∗)

Variables

- long **REFRESH_COUNTER**

6.3.1 Function Documentation

6.3.1.1 calculate_positions()

```
void calculate_positions (
             int ∗ x,
             int ∗ y )
```

Generate x and y coordinates for ncurses print

Parameters

x	array for x values
y	array for y values

Definition at line 104 of file generic.c.

6.3.1.2 create_hash()

```
uint32_t create_hash (
             int ∗ array )
```

Create hash from array

Parameters

array	array

Definition at line 121 of file generic.c.

6.3.1.3 file_not_found()

```
int file_not_found (
             FILE * src,
             char * name )
```

Called, when file is not found

Parameters

src	file
name	file name

Definition at line 66 of file generic.c.

6.3.1.4 file_size()

```
long file_size (
             FILE * src )
```

Get file size, then rewind

Parameters

src	file

Definition at line 93 of file generic.c.

6.3.1.5 help()

```
void help (
             void )
```

Print help

Definition at line 11 of file generic.c.

6.3.1.6 is_empty()

```
int is_empty (
            FILE * src )
```

Check, if file is empty

Parameters

src	file

Definition at line 78 of file generic.c.

6.3.1.7 print_array_ncurses()

```
void print_array_ncurses (
            WINDOW * win,
            int * array,
            int * positions_y,
            int * positions_x )
```

Print array via ncurses liblary

Parameters

win	ncurses window
array	values to print
positions↩ _y	values for printing in y axis
positions↩ _x	values for printing in x axis

Definition at line 141 of file generic.c.

6.3.1.8 print_ncurses()

```
void print_ncurses (
            WINDOW * win,
            int * array,
            int * positions_y,
            int * positions_x )
```

Call print_array_nucrses every REFRESH_INTERVAL calls

Parameters

win	ncurses window

Parameters

array	values to print
positions↩_y	values for printing in y axis
positions↩_x	values for printing in x axis

Definition at line 159 of file generic.c.

6.3.1.9 print_to_file()

```
int print_to_file (
            char * str,
            int * array )
```

Print array to file

Parameters

str	file name
array	array of values

Definition at line 23 of file generic.c.

6.3.1.10 read_array_from_file()

```
void read_array_from_file (
            FILE * source,
            int * array_before,
            int * array_after )
```

Load array from CSV file

Parameters

source	file
array_before	array before solving
array_after	array after solving

Definition at line 39 of file generic.c.

6.4 src/json.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "cJSON/cJSON.h"
#include "generic.h"
#include "param.h"
#include "json.h"
```

Functions

- char ∗ json_create_missing (void)
- char ∗ json_create_object (int ∗array_before, int ∗array_after, uint32_t hash, char ∗json_string, long size)
- int json_compare_hash (char ∗string, long len, uint32_t hash)
- int json_find_existing (char ∗string, long len, uint32_t hash, int ∗grid_after)
- int json_found_problem (void)
- int json_check_integrity (char ∗string, long len)
- int json_history (char ∗string, long len)
- char ∗ start_history (arg_t ∗head, long ∗size)
- void start_previous (arg_t ∗head, char ∗string, long fsize)

6.4.1 Function Documentation

6.4.1.1 json_check_integrity()

```
int json_check_integrity (
            char * string,
            long len )
```

Check, if JSON file is valid

Parameters

string	contents of JSON file
len	size of JSON file

Definition at line 141 of file json.c.

6.4.1.2 json_compare_hash()

```
int json_compare_hash (
            char * string,
            long len,
            uint32_t hash )
```

Check, if entry already exists

Parameters

string	contents of JSON file
len	size of JSON file
hash	hash

Definition at line 77 of file json.c.

6.4.1.3 json_create_missing()

```
char* json_create_missing (
            void )
```

Create empty JSON objects

Definition at line 12 of file json.c.

6.4.1.4 json_create_object()

```
char* json_create_object (
            int * array_before,
            int * array_after,
            uint32_t hash,
            char * json_string,
            long size )
```

Add entry for puzlle into JSON

Parameters

array_before	array before solving
array_after	array after solving
hash	hash
json_string	contents of JSON
size	size of JSON string

Definition at line 33 of file json.c.

6.4.1.5 json_find_existing()

```
int json_find_existing (
            char * string,
            long len,
            uint32_t hash,
            int * grid_after )
```

Get object form JSON

Parameters

string	contents of JSON file
len	size of JSON file
hash	hash
grid_after	array after_solving

Definition at line 98 of file json.c.

6.4.1.6 json_found_problem()

```
int json_found_problem (
            void )
```

Called, when problem occured with JSON file

Definition at line 125 of file json.c.

6.4.1.7 json_history()

```
int json_history (
            char * string,
            long len )
```

Iterate through JSON file

Parameters

string	contents of JSON file
len	size of JSON file

Definition at line 190 of file json.c.

6.4.1.8 start_history()

```
char* start_history (
            arg_t * head,
            long * size )
```

Initialize JSON

Parameters

head	head of struct
size	size of JSON file

Definition at line 306 of file json.c.

6.4.1.9 start_previous()

```
void start_previous (
            arg_t * head,
            char * string,
            long fsize )
```

Previous function

Parameters

head	head of struct
string	contents of JSON file
fsize	size of JSON file

Definition at line 351 of file json.c.

6.5 src/json.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "cJSON/cJSON.h"
#include "generic.h"
#include "param.h"
```

Functions

- char ∗ json_create_missing (void)
- char ∗ json_create_object (int ∗, int ∗, uint32_t, char ∗, long)

- int json_compare_hash (char ∗, long, uint32_t)
- int json_find_existing (char ∗, long, uint32_t, int ∗)
- int json_found_problem (void)
- int json_check_integrity (char ∗, long)
- int json_history (char ∗, long)
- char ∗ start_history (arg_t ∗, long ∗)
- void start_previous (arg_t ∗, char ∗, long)

6.5.1 Function Documentation

6.5.1.1 json_check_integrity()

```
int json_check_integrity (
            char * string,
            long len )
```

Check, if JSON file is valid

Parameters

string	contents of JSON file
len	size of JSON file

Definition at line 141 of file json.c.

6.5.1.2 json_compare_hash()

```
int json_compare_hash (
            char * string,
            long len,
            uint32_t hash )
```

Check, if entry already exists

Parameters

string	contents of JSON file
len	size of JSON file
hash	hash

Definition at line 77 of file json.c.

6.5.1.3 json_create_missing()

```
char* json_create_missing (
            void )
```

Create empty JSON objects

Definition at line 12 of file json.c.

6.5.1.4 json_create_object()

```
char* json_create_object (
            int * array_before,
            int * array_after,
            uint32_t hash,
            char * json_string,
            long size )
```

Add entry for puzlle into JSON

Parameters

array_before	array before solving
array_after	array after solving
hash	hash
json_string	contents of JSON
size	size of JSON string

Definition at line 33 of file json.c.

6.5.1.5 json_find_existing()

```
int json_find_existing (
            char * string,
            long len,
            uint32_t hash,
            int * grid_after )
```

Get object form JSON

Parameters

string	contents of JSON file
len	size of JSON file
hash	hash
grid_after	array after_solving

Definition at line 98 of file json.c.

6.5.1.6 json_found_problem()

```
int json_found_problem (
            void )
```

Called, when problem occured with JSON file

Definition at line 125 of file json.c.

6.5.1.7 json_history()

```
int json_history (
            char * string,
            long len )
```

Iterate through JSON file

Parameters

string	contents of JSON file
len	size of JSON file

Definition at line 190 of file json.c.

6.5.1.8 start_history()

```
char* start_history (
            arg_t * head,
            long * size )
```

Initialize JSON

Parameters

head	head of struct
size	size of JSON file

Definition at line 306 of file json.c.

6.5.1.9 start_previous()

```
void start_previous (
             arg_t * head,
             char * string,
             long fsize )
```

Previous function

Parameters

head	head of struct
string	contents of JSON file
fsize	size of JSON file

Definition at line 351 of file json.c.

6.6 src/param.c File Reference

```
#include <stdlib.h>
#include "generic.h"
#include "param.h"
```

Functions

- void add_to_end (arg_t ∗head)
- arg_t ∗ param_create_struct (void)
- void param_disable_history (arg_t ∗head)
- void param_destroy_struct (arg_t ∗head)
- int param_status_input (arg_t ∗head)
- int param_status_output (arg_t ∗head)
- int param_status_history (arg_t ∗head)
- int param_status_force (arg_t ∗head)
- int param_status_previous (arg_t ∗head)
- void param_create_input (arg_t ∗head, int argv_index)
- void param_create_output (arg_t ∗head, int argv_index)
- void param_create_force (arg_t ∗head)
- void param_create_previous (arg_t ∗head)

6.6.1 Function Documentation

6.6.1.1 add_to_end()

```
void add_to_end (
             arg_t * head )
```

Allocate memory at the end of list

Parameters

head	head of struct

Definition at line 9 of file param.c.

6.6.1.2 param_create_force()

```
void param_create_force (
             arg_t * head )
```

Create entry for force flag

Parameters

head	head of struct

Definition at line 162 of file param.c.

6.6.1.3 param_create_input()

```
void param_create_input (
             arg_t * head,
             int argv_index )
```

Create entry for input flag

Parameters

head	head of struct
argv_index	argv index

Definition at line 128 of file param.c.

6.6.1.4 param_create_output()

```
void param_create_output (
             arg_t * head,
             int argv_index )
```

Create entry for output flag

Parameters

head	head of struct
argv_index	argv index

Definition at line 145 of file param.c.

6.6.1.5 param_create_previous()

```
void param_create_previous (
            arg_t * head )
```

Create entry for previous flag

Parameters

head	head of struct

Definition at line 179 of file param.c.

6.6.1.6 param_create_struct()

```
arg_t* param_create_struct (
            void  )
```

Create first element of list

Definition at line 20 of file param.c.

6.6.1.7 param_destroy_struct()

```
void param_destroy_struct (
            arg_t * head )
```

Delete list

Parameters

head	head of struct

Definition at line 46 of file param.c.

6.6.1.8 param_disable_history()

```
void param_disable_history (
            arg_t * head )
```

Disable history file

Parameters

head	head of struct

Definition at line 33 of file param.c.

6.6.1.9 param_status_force()

```
int param_status_force (
            arg_t * head )
```

Check, if force is activated

Parameters

head	head of struct

Definition at line 99 of file param.c.

6.6.1.10 param_status_history()

```
int param_status_history (
            arg_t * head )
```

Check, if history is activated

Parameters

head	head of struct

Definition at line 85 of file param.c.

6.6.1.11 param_status_input()

```
int param_status_input (
            arg_t * head )
```

Check, if input is activated

Parameters

head	head of struct

Definition at line 57 of file param.c.

6.6.1.12 param_status_output()

```
int param_status_output (
            arg_t * head )
```

Check, if output is activated

Parameters

head	head of struct

Definition at line 71 of file param.c.

6.6.1.13 param_status_previous()

```
int param_status_previous (
            arg_t * head )
```

Check, if previous is activated

Parameters

head	head of struct

Definition at line 113 of file param.c.

6.7 src/param.h File Reference

```
#include <stdlib.h>
#include "generic.h"
```

Macros

- #define **stop_program**(head, string) param_destroy_struct(head); if(string != NULL){free(string); string = NULL;}; return 0;

Functions

- void add_to_end (arg_t ∗)
- arg_t ∗ param_create_struct (void)
- void param_disable_history (arg_t ∗)
- void param_destroy_struct (arg_t ∗)
- int param_status_input (arg_t ∗)
- int param_status_output (arg_t ∗)
- int param_status_history (arg_t ∗)
- int param_status_force (arg_t ∗)
- int param_status_previous (arg_t ∗)
- void param_create_input (arg_t ∗, int)
- void param_create_output (arg_t ∗, int)
- void param_create_force (arg_t ∗)
- void param_create_previous (arg_t ∗)

6.7.1 Function Documentation

6.7.1.1 add_to_end()

```
void add_to_end (
             arg_t * head )
```

Allocate memory at the end of list

Parameters

head	head of struct

Definition at line 9 of file param.c.

6.7.1.2 param_create_force()

```
void param_create_force (
             arg_t * head )
```

Create entry for force flag

Parameters

head	head of struct

Definition at line 162 of file param.c.

6.7.1.3 param_create_input()

```
void param_create_input (
            arg_t * head,
            int argv_index )
```

Create entry for input flag

Parameters

head	head of struct
argv_index	argv index

Definition at line 128 of file param.c.

6.7.1.4 param_create_output()

```
void param_create_output (
            arg_t * head,
            int argv_index )
```

Create entry for output flag

Parameters

head	head of struct
argv_index	argv index

Definition at line 145 of file param.c.

6.7.1.5 param_create_previous()

```
void param_create_previous (
            arg_t * head )
```

Create entry for previous flag

Parameters

head	head of struct

Definition at line 179 of file param.c.

6.7.1.6 param_create_struct()

```
arg_t* param_create_struct (
             void  )
```

Create first element of list

Definition at line 20 of file param.c.

6.7.1.7 param_destroy_struct()

```
void param_destroy_struct (
             arg_t * head )
```

Delete list

Parameters

head	head of struct

Definition at line 46 of file param.c.

6.7.1.8 param_disable_history()

```
void param_disable_history (
             arg_t * head )
```

Disable history file

Parameters

head	head of struct

Definition at line 33 of file param.c.

6.7.1.9 param_status_force()

```
int param_status_force (
             arg_t * head )
```

Check, if force is activated

Parameters

head	head of struct

Definition at line 99 of file param.c.

6.7.1.10 param_status_history()

```
int param_status_history (
            arg_t * head )
```

Check, if history is activated

Parameters

head	head of struct

Definition at line 85 of file param.c.

6.7.1.11 param_status_input()

```
int param_status_input (
            arg_t * head )
```

Check, if input is activated

Parameters

head	head of struct

Definition at line 57 of file param.c.

6.7.1.12 param_status_output()

```
int param_status_output (
            arg_t * head )
```

Check, if output is activated

Parameters

head	head of struct

Definition at line 71 of file param.c.

6.7.1.13 param_status_previous()

```
int param_status_previous (
            arg_t * head )
```

Check, if previous is activated

Parameters

head	head of struct

Definition at line 113 of file param.c.

6.8 src/solving.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <ncurses.h>
#include <time.h>
#include "generic.h"
#include "param.h"
#include "json.h"
#include "solving.h"
```

Functions

- int sudoku_checker (int ∗array, int x, int y, int value)
- int sudoku_validator (int ∗array)
- int sudoku_solver (WINDOW ∗win, int ∗array, int ∗positions_y, int ∗positions_x, int y, int x)
- int start_solver (arg_t ∗head, char ∗string, long fsize, char ∗argv[ ])

6.8.1 Function Documentation

6.8.1.1 start_solver()

```
int start_solver (
            arg_t * head,
            char * string,
            long fsize,
            char * argv[ ] )
```

Start solving

Parameters

head	head of struct
string	contents of JSON file
fsize	size of JSON file
argv[ ]	argv array

Definition at line 131 of file solving.c.

6.8.1.2 sudoku_checker()

```
int sudoku_checker (
             int * array,
             int x,
             int y,
             int value )
```

Check, if provided value can be inserted

Parameters

array	array of values
x	x position
y	y position
value	value to check

Definition at line 18 of file solving.c.

6.8.1.3 sudoku_solver()

```
int sudoku_solver (
             WINDOW * win,
             int * array,
             int * positions_y,
             int * positions_x,
             int y,
             int x )
```

Main solving function

Parameters

win	ncurses window
arrray	array of values
positions←_y	coordinates for printing in y axis
positions←_x	coordinates for printing in x axis
y	position y
x	position x

Definition at line 86 of file solving.c.

6.8.1.4 sudoku_validator()

```
int sudoku_validator (
            int * array )
```

Check, if there are no contradictions in input file

Parameters

array	array of values

Definition at line 65 of file solving.c.

6.9 src/solving.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <ncurses.h>
#include <time.h>
#include "generic.h"
#include "param.h"
#include "json.h"
```

Functions

- int sudoku_checker (int ∗, int, int, int)
- int sudoku_validator (int ∗)
- int sudoku_solver (WINDOW ∗, int ∗, int ∗, int ∗, int, int)
- int start_solver (arg_t ∗, char ∗, long, char ∗[ ])

6.9.1 Function Documentation

6.9.1.1 start_solver()

```
int start_solver (
            arg_t * head,
            char * string,
            long fsize,
            char * argv[ ] )
```

Start solving

Parameters

head	head of struct
string	contents of JSON file
fsize	size of JSON file
argv[ ]	argv array

Definition at line 131 of file solving.c.

6.9.1.2 sudoku_checker()

```
int sudoku_checker (
            int * array,
            int x,
            int y,
            int value )
```

check, if provided value can be inserted

Check, if provided value can be inserted

Parameters

array	array of values
x	x position
y	y position
value	value to check

Definition at line 18 of file solving.c.

6.9.1.3 sudoku_solver()

```
int sudoku_solver (
            WINDOW * win,
            int * array,
            int * positions_y,
            int * positions_x,
            int y,
            int x )
```

Main solving function

Parameters

win	ncurses window
arrray	array of values

Parameters

positions_y	coordinates for printing in y axis
positions_x	coordinates for printing in x axis
y	position y
x	position x

Definition at line 86 of file solving.c.

6.9.1.4 sudoku_validator()

```
int sudoku_validator (
            int * array )
```

check, if there are no contradictions in input file

Check, if there are no contradictions in input file

Parameters

array	array of values

Definition at line 65 of file solving.c.