libconfig++

Learning objectives

Use of applications configuration file

Access in C to configuration files

Libconfig++ is a library for reading structured configuration files. The use of this library relieves the programmer from processing and parsing files containing application configuration settings, thus allowing the run-time configuration of applications.

1 Installation

To use this library it is necessary to install it on Linux or mac OS X.

1.1 Linux

The **libcofig++** installation in Linux is straightforward and depend on the Linux version.

For Debina/Ubuntu it is necessary to issue the following commands

```
sudo apt install libconfig++-dev
```

1.2 MAC OS X

To install the SDL2 library in MAC OS X it is necessary to use the **brew** package manager:

https://brew.sh/

after installing brew it is necessary to type the following command in the terminal:

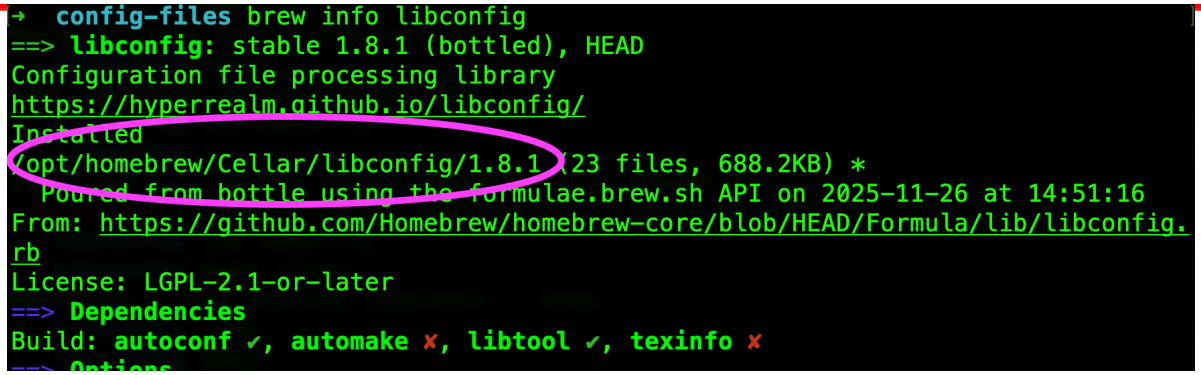
```
brew install libconfig
```

MAC OS X

In Mac OS X, brew installs libraries in non default directories, and as such gcc does not automaticaly finds the .h files and the libraries.

1 – Verify where the libconfig++ files were installed running the command

brew info libconfig



2 – Use the installation directory when compiling your application and in the makefile:

```
gcc config_test.c -o config_test \
     -lconfig++ \
     -I/opt/homebrew/Cellar/libconfig/1.8.1/include \
     -L/opt/homebrew/Cellar/libconfig/1.8.1/lib
```

If the package was installed in a different directory the command **brew info sdl2** will print that location.

To help Mac OS X students we provide an example makefile.

2 Programming model

https://hyperrealm.github.io/libconfig/libconfig_manual.html#The-C-API

The libconfig++ allows the access to data stored in configuration files. This library has a function to open a configuration file and other functions to retrieve values of different types.

2.1 Configuration files

The simples type of configuration files handled by libconfig++ consist of lines with the name and values of configuration variables (called settings). The setting have this format:

```
name = value ;
```

or:

```
name : value ;
```

The trailing semicolon is optional. Whitespace is not significant. The value may be a scalar value, an array, a group, or a list.

The provided example show settings of three different types (int, float and string):

```
parametro_1_int = 30
parametro_2_float = 0.5
parametro_3_string = "xpto"
```

2.2 Compilation

To correctly compile a program that users **libconfig++** it is necessary to do the next include:

```
#include <libconfig.h>
```

When linking it is necessary to add the **-lconfig++** parammetr to gcc:

```
gcc config_test.c -o config_test -lconfig++
```

2.3 Programming

To read the settings stored in a configuration file, this library has a function that opens such configuration file, parses the various settings and stores these values in memory. Then the programmer can access the variopus stored values through their name and type, as exemplified in the supplied code:

`config_t cfg;`
 	
`config_init(&cfg);`
	Declaration of the varriable that will hold the settings
	
	Initialization of the libconfig++ library.

`if(!config_read_file(&cfg, "example.conf")) {`
`  fprintf(stderr, "error\n");`
`  config_destroy(&cfg);`
`  return 1;`
`}`
	Definition of the configuration file name, access and parse of the settings

`int parametro_1_int;`
`config_lookup_int(&cfg,`
`  "parametro_i_int",`
`  &parametro_1_int);`
`printf("Max parametro_1_float: %d\n",`
`  parametro_1_int);`
	Each type of setting (int, float/double or string) has his own functions.
	
	The **config_lookup_int** receives as argument the cfg variable, the name of the setting and a pointer to the interger that will hold the setting value.

`int missing_parameter;`
`if(config_lookup_int(&cfg,`
`  "miss_param",`
`  &miss_param)== CONFIG_FALSE){`
`  printf("Configuration not found\n");`
`}`
	It is possible that the programmer uses an incorrect setting name or there is a mismatch with the type of the setting. In these cases the lookup function returns **CONFIG_FALSE**

`double parametro_2_float;`
`config_lookup_float(&cfg,`
`  "parametro_2_float",`
`  &parametro_2_float);`
`printf("parametro_2_float: %f\n",`
`  parametro_2_float);`
	The **config_lookup_float** receives as argument the cfg variable, the name of the setting and a pointer to the double that will hold the setting value.

`char * parametro_3_string;`
`config_lookup_string(&cfg,`
`"parametro_3_string",`
`&parametro_3_string);`
`printf("parametro_3_string: %s\n",`
`parametro_3_string);`
	The **config_lookup_string** receives as argument the cfg variable, the name of the setting and a pointer to a pointer to char. This function manages the memory that will hold the string and as such the programmer should not free it.

`config_destroy(&cfg);`	After accessing all settings the configuration file can be closed