

Lab3_gr2

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

participantes	??
-------------------------------	-------	----

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

build/CMakeFiles/3.25.0/CompilerIdC/CMakeCCompilerId.c	??
build/CMakeFiles/3.25.0/CompilerIdCXX/CMakeCXXCompilerId.cpp	??
build/CMakeFiles/labpp.bin.dir/src/config_tty.cpp.o.d	??
build/CMakeFiles/labpp.bin.dir/src/main.cpp.o.d	??
build/CMakeFiles/labpp.bin.dir/src/participant.cpp.o.d	??
CMakeFiles/3.25.0/CompilerIdC/CMakeCCompilerId.c	??
CMakeFiles/3.25.0/CompilerIdCXX/CMakeCXXCompilerId.cpp	??
CMakeFiles/labpp.bin.dir/src/config_tty.cpp.o.d	??
CMakeFiles/labpp.bin.dir/src/main.cpp.o.d	??
CMakeFiles/labpp.bin.dir/src/participant.cpp.o.d	??
include/lib_grupo2.h	??
include/tty_lib2.h	??
src/config_tty.cpp	??
src/main.cpp	??
src/participant.cpp	??

Chapter 3

Class Documentation

3.1 participantes Class Reference

```
#include <lib_grupo2.h>
```

Public Member Functions

- [participantes](#) ()
- [participantes](#) (unsigned int participante_id, string part_name)
- void [set_participantes](#) ()
- void [set_pushed](#) (unsigned int cant)
- unsigned int [get_participante_id](#) (unsigned int participante_id)
- string [get_name](#) (string part_name)
- unsigned int [get_pushed](#) (unsigned int num_pushed)
- unsigned int [get_ImprimirInfo](#) (unsigned int [ImprimirInfo](#))
- void [ImprimirInfo](#) ()

3.1.1 Detailed Description

Definition at line 11 of file lib_grupo2.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 participantes() [1/2]

```
participantes::participantes ( )
```

declaracion de los constructores inicializacion de las variables y funciones para la clase participantes

Definition at line 10 of file participant.cpp.

```
10 :participante_id(0),part_name(" "),num_pushed(0){}
```

3.1.2.2 participantes() [2/2]

```
participantes::participantes (
    unsigned int participante_id,
    string part_name )
```

Definition at line 11 of file participant.cpp.

```
11 :participante_id(id),part_name(name),num_pushed(0){}
```

3.1.3 Member Function Documentation

3.1.3.1 get_ImprimirInfo()

```
unsigned int participantes::get_ImprimirInfo (
    unsigned int ImprimirInfo )
```

3.1.3.2 get_name()

```
string participantes::get_name (
    string part_name )
```

Definition at line 26 of file participant.cpp.

```
26                                     {
27     return part_name;
28 }
```

3.1.3.3 get_participante_id()

```
unsigned int participantes::get_participante_id (
    unsigned int participante_id )
```

Definition at line 23 of file participant.cpp.

```
23                                     {
24     return participante_id;
25 }
```

3.1.3.4 get_pushed()

```
unsigned int participantes::get_pushed (
    unsigned int num_pushed )
```

Definition at line 30 of file participant.cpp.

```
30                                     {
31     return num_pushed;
32 }
```

3.1.3.5 ImprimirInfo()

```
void participantes::ImprimirInfo ( )
```

Definition at line 35 of file participant.cpp.

```
36 {
37     cout <<"-----" <<'\\n';
38     cout <<"El nombre del participante es: " <<part_name<<'\\n';
39     cout <<"El ID participante es: " << participante_id<<'\\n';
40     cout <<"El # de pulsaciones que tuvo es: " <<num_pushed <<'\\n';
41     cout <<"-----" <<'\\n';
42 }
```

3.1.3.6 set_participantes()

```
void participantes::set_participantes ( )
```

Definition at line 12 of file participant.cpp.

```
12                                     {
13     cout << "ingrese el id del participantes" << endl;
14     cin >> participante_id;
15
16     cout << "ingrese el nombre del participante:" << endl;
17     cin >> part_name;
18 }
```

3.1.3.7 set_pushed()

```
void participantes::set_pushed (
    unsigned int cant )
```

Definition at line 19 of file participant.cpp.

```
19                                     {
20     num_pushed =cant;
21 }
```

The documentation for this class was generated from the following files:

- [include/lib_grupo2.h](#)
- [src/participant.cpp](#)

Chapter 4

File Documentation

4.1 build/CMakeFiles/3.25.0/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- `#define __has_include(x) 0`
- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define C_VERSION`

Functions

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

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"`
- `char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"`
- `char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"`
- `const char * info_language_standard_default`
- `const char * info_language_extensions_default`

4.1.1 Macro Definition Documentation

4.1.1.1 __has_include

```
#define __has_include(  
    x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

4.1.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 718 of file CMakeCCompilerId.c.

4.1.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 807 of file CMakeCCompilerId.c.

4.1.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 429 of file CMakeCCompilerId.c.

4.1.1.5 DEC

```
#define DEC(  
    n )
```

Value:

```
('0' + (((n) / 10000000) % 10)), \
('0' + (((n) / 1000000) % 10)), \
('0' + (((n) / 100000) % 10)), \
('0' + (((n) / 10000) % 10)), \
('0' + (((n) / 1000) % 10)), \
('0' + (((n) / 100) % 10)), \
('0' + (((n) / 10) % 10)), \
('0' + ((n) % 10))
```

Definition at line 722 of file CMakeCCompilerId.c.

4.1.1.6 HEX

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \  
('0' + ((n)>>24 & 0xF)), \  
('0' + ((n)>>20 & 0xF)), \  
('0' + ((n)>>16 & 0xF)), \  
('0' + ((n)>>12 & 0xF)), \  
('0' + ((n)>>8  & 0xF)), \  
('0' + ((n)>>4  & 0xF)), \  
('0' + ((n)      & 0xF))
```

Definition at line 733 of file CMakeCCompilerId.c.

4.1.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 560 of file CMakeCCompilerId.c.

4.1.1.8 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY\_HELPER(X)
```

Definition at line 450 of file CMakeCCompilerId.c.

4.1.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 449 of file CMakeCCompilerId.c.

4.1.2 Function Documentation

4.1.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Definition at line 841 of file CMakeCCompilerId.c.

```
843 {
844     int require = 0;
845     require += info_compiler[argc];
846     require += info_platform[argc];
847     require += info_arch[argc];
848     #ifdef COMPILER_VERSION_MAJOR
849     require += info_version[argc];
850     #endif
851     #ifdef COMPILER_VERSION_INTERNAL
852     require += info_version_internal[argc];
853     #endif
854     #ifdef SIMULATE_ID
855     require += info_simulate[argc];
856     #endif
857     #ifdef SIMULATE_VERSION_MAJOR
858     require += info_simulate_version[argc];
859     #endif
860     #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
861     require += info_cray[argc];
862     #endif
863     require += info_language_standard_default[argc];
864     require += info_language_extensions_default[argc];
865     (void)argv;
866     return require;
867 }
```

4.1.3 Variable Documentation

4.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 799 of file CMakeCCompilerId.c.

4.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 436 of file CMakeCCompilerId.c.

4.1.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["  
  
    "OFF"  
    "]"
```

Definition at line 823 of file CMakeCCompilerId.c.

4.1.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=  
    "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 820 of file CMakeCCompilerId.c.

4.1.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 798 of file CMakeCCompilerId.c.

4.2 CMakeFiles/3.25.0/CompilerIdC/CompilerId.c File Reference

Macros

- `#define __has_include(x) 0`
- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define C_VERSION`

Functions

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

Variables

- char const * [info_compiler](#) = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * [info_platform](#) = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * [info_arch](#) = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * [info_language_standard_default](#)
- const char * [info_language_extensions_default](#)

4.2.1 Macro Definition Documentation

4.2.1.1 __has_include

```
#define __has_include(  
    x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

4.2.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 718 of file CMakeCCompilerId.c.

4.2.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 807 of file CMakeCCompilerId.c.

4.2.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 429 of file CMakeCCompilerId.c.

4.2.1.5 DEC

```
#define DEC(  
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \
('0' + ((n) / 1000000) % 10), \
('0' + ((n) / 100000) % 10), \
('0' + ((n) / 10000) % 10), \
('0' + ((n) / 1000) % 10), \
('0' + ((n) / 100) % 10), \
('0' + ((n) / 10) % 10), \
('0' + ((n) % 10))
```

Definition at line 722 of file CMakeCCompilerId.c.

4.2.1.6 HEX

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \
('0' + ((n) >> 24 & 0xF)), \
('0' + ((n) >> 20 & 0xF)), \
('0' + ((n) >> 16 & 0xF)), \
('0' + ((n) >> 12 & 0xF)), \
('0' + ((n) >> 8 & 0xF)), \
('0' + ((n) >> 4 & 0xF)), \
('0' + ((n) & 0xF))
```

Definition at line 733 of file CMakeCCompilerId.c.

4.2.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 560 of file CMakeCCompilerId.c.

4.2.1.8 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY_HELPER(X)
```

Definition at line 450 of file CMakeCCompilerId.c.

4.2.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 449 of file CMakeCCompilerId.c.

4.2.2 Function Documentation

4.2.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 841 of file CMakeCCompilerId.c.

```
843 {  
844     int require = 0;  
845     require += info_compiler[argc];  
846     require += info_platform[argc];  
847     require += info_arch[argc];  
848 #ifdef COMPILER_VERSION_MAJOR  
849     require += info_version[argc];  
850 #endif  
851 #ifdef COMPILER_VERSION_INTERNAL  
852     require += info_version_internal[argc];  
853 #endif  
854 #ifdef SIMULATE_ID  
855     require += info_simulate[argc];  
856 #endif  
857 #ifdef SIMULATE_VERSION_MAJOR  
858     require += info_simulate_version[argc];  
859 #endif  
860 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)  
861     require += info_cray[argc];  
862 #endif  
863     require += info_language_standard_default[argc];  
864     require += info_language_extensions_default[argc];  
865     (void)argv;  
866     return require;  
867 }
```

4.2.3 Variable Documentation

4.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 799 of file CMakeCCompilerId.c.

4.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 436 of file CMakeCCompilerId.c.

4.2.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
```

```
"OFF"
```

```
"]"
```

Definition at line 823 of file CMakeCCompilerId.c.

4.2.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=  
"INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 820 of file CMakeCCompilerId.c.

4.2.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 798 of file CMakeCCompilerId.c.

4.3 build/CMakeFiles/3.25.0/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- `#define __has_include(x) 0`
- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define CXX_STD __cplusplus`

Functions

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

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"`
- `char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"`
- `char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"`
- `const char * info_language_standard_default`
- `const char * info_language_extensions_default`

4.3.1 Macro Definition Documentation

4.3.1.1 __has_include

```
#define __has_include(  
    x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

4.3.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 703 of file CMakeCXXCompilerId.cpp.

4.3.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 414 of file CMakeCXXCompilerId.cpp.

4.3.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 801 of file CMakeCXXCompilerId.cpp.

4.3.1.5 DEC

```
#define DEC(  
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \
('0' + ((n) / 1000000) % 10), \
('0' + ((n) / 100000) % 10), \
('0' + ((n) / 10000) % 10), \
('0' + ((n) / 1000) % 10), \
('0' + ((n) / 100) % 10), \
('0' + ((n) / 10) % 10), \
('0' + ((n) % 10))
```

Definition at line 707 of file CMakeCXXCompilerId.cpp.

4.3.1.6 HEX

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \
('0' + ((n) >> 24 & 0xF)), \
('0' + ((n) >> 20 & 0xF)), \
('0' + ((n) >> 16 & 0xF)), \
('0' + ((n) >> 12 & 0xF)), \
('0' + ((n) >> 8 & 0xF)), \
('0' + ((n) >> 4 & 0xF)), \
('0' + ((n) & 0xF))
```

Definition at line 718 of file CMakeCXXCompilerId.cpp.

4.3.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 545 of file CMakeCXXCompilerId.cpp.

4.3.1.8 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY\_HELPER(X)
```

Definition at line 435 of file CMakeCXXCompilerId.cpp.

4.3.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 434 of file CMakeCXXCompilerId.cpp.

4.3.2 Function Documentation

4.3.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 832 of file CMakeCXXCompilerId.cpp.

```
833 {  
834     int require = 0;  
835     require += info\_compiler[argc];  
836     require += info\_platform[argc];  
837     require += info\_arch[argc];  
838     #ifdef COMPILER_VERSION_MAJOR  
839     require += info\_version[argc];  
840     #endif  
841     #ifdef COMPILER_VERSION_INTERNAL  
842     require += info\_version\_internal[argc];  
843     #endif  
844     #ifdef SIMULATE_ID  
845     require += info\_simulate[argc];  
846     #endif  
847     #ifdef SIMULATE_VERSION_MAJOR  
848     require += info\_simulate\_version[argc];  
849     #endif  
850     #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)  
851     require += info\_cray[argc];  
852     #endif  
853     require += info\_language\_standard\_default[argc];  
854     require += info\_language\_extensions\_default[argc];  
855     (void)argv;  
856     return require;  
857 }
```


4.3.3 Variable Documentation

4.3.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 784 of file CMakeCXXCompilerId.cpp.

4.3.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 421 of file CMakeCXXCompilerId.cpp.

4.3.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
```

```
"OFF"
```

```
"]"
```

Definition at line 820 of file CMakeCXXCompilerId.cpp.

4.3.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
```

```
"98"
```

```
"]"
```

Definition at line 804 of file CMakeCXXCompilerId.cpp.

4.3.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 783 of file CMakeCXXCompilerId.cpp.

4.4 CMakeFiles/3.25.0/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- `#define __has_include(x) 0`
- `#define COMPILER_ID ""`
- `#define STRINGIFY_HELPER(X) #X`
- `#define STRINGIFY(X) STRINGIFY_HELPER(X)`
- `#define PLATFORM_ID`
- `#define ARCHITECTURE_ID`
- `#define DEC(n)`
- `#define HEX(n)`
- `#define CXX_STD __cplusplus`

Functions

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

Variables

- `char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"`
- `char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"`
- `char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"`
- `const char * info_language_standard_default`
- `const char * info_language_extensions_default`

4.4.1 Macro Definition Documentation

4.4.1.1 __has_include

```
#define __has_include(  
    x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

4.4.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 703 of file CMakeCXXCompilerId.cpp.

4.4.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 414 of file CMakeCXXCompilerId.cpp.

4.4.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 801 of file CMakeCXXCompilerId.cpp.

4.4.1.5 DEC

```
#define DEC(  
    n )
```

Value:

```
('0' + ((n) / 10000000) % 10), \
('0' + ((n) / 1000000) % 10), \
('0' + ((n) / 100000) % 10), \
('0' + ((n) / 10000) % 10), \
('0' + ((n) / 1000) % 10), \
('0' + ((n) / 100) % 10), \
('0' + ((n) / 10) % 10), \
('0' + ((n) % 10))
```

Definition at line 707 of file CMakeCXXCompilerId.cpp.

4.4.1.6 HEX

```
#define HEX(  
    n )
```

Value:

```
('0' + ((n) >> 28 & 0xF)), \
('0' + ((n) >> 24 & 0xF)), \
('0' + ((n) >> 20 & 0xF)), \
('0' + ((n) >> 16 & 0xF)), \
('0' + ((n) >> 12 & 0xF)), \
('0' + ((n) >> 8 & 0xF)), \
('0' + ((n) >> 4 & 0xF)), \
('0' + ((n) & 0xF))
```

Definition at line 718 of file CMakeCXXCompilerId.cpp.

4.4.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 545 of file CMakeCXXCompilerId.cpp.

4.4.1.8 STRINGIFY

```
#define STRINGIFY(  
    X )  STRINGIFY_HELPER(X)
```

Definition at line 435 of file CMakeCXXCompilerId.cpp.

4.4.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 434 of file CMakeCXXCompilerId.cpp.

4.4.2 Function Documentation

4.4.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 832 of file CMakeCXXCompilerId.cpp.

```
833 {  
834     int require = 0;  
835     require += info_compiler[argc];  
836     require += info_platform[argc];  
837     require += info_arch[argc];  
838     #ifdef COMPILER_VERSION_MAJOR  
839     require += info_version[argc];  
840     #endif  
841     #ifdef COMPILER_VERSION_INTERNAL  
842     require += info_version_internal[argc];  
843     #endif  
844     #ifdef SIMULATE_ID  
845     require += info_simulate[argc];  
846     #endif  
847     #ifdef SIMULATE_VERSION_MAJOR  
848     require += info_simulate_version[argc];  
849     #endif  
850     #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)  
851     require += info_cray[argc];  
852     #endif  
853     require += info_language_standard_default[argc];  
854     require += info_language_extensions_default[argc];  
855     (void)argv;  
856     return require;  
857 }
```

4.4.3 Variable Documentation

4.4.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 784 of file CMakeCXXCompilerId.cpp.

4.4.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 421 of file CMakeCXXCompilerId.cpp.

4.4.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
```

```
    "OFF"
```

```
"]"
```

Definition at line 820 of file CMakeCXXCompilerId.cpp.

4.4.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
```

```
    "98"
```

```
"]"
```

Definition at line 804 of file CMakeCXXCompilerId.cpp.

4.4.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 783 of file CMakeCXXCompilerId.cpp.

4.5 build/CMakeFiles/labpp.bin.dir/src/config_tty.cpp.o.d File Reference

4.6 CMakeFiles/labpp.bin.dir/src/config_tty.cpp.o.d File Reference

4.7 build/CMakeFiles/labpp.bin.dir/src/main.cpp.o.d File Reference

4.8 CMakeFiles/labpp.bin.dir/src/main.cpp.o.d File Reference

4.9 build/CMakeFiles/labpp.bin.dir/src/participant.cpp.o.d File Reference

4.10 CMakeFiles/labpp.bin.dir/src/participant.cpp.o.d File Reference

4.11 include/lib_grupo2.h File Reference

```
#include <iostream>
#include <string.h>
#include <unistd.h>
#include "tty_lib2.h"
```

Classes

- class [participantes](#)

4.12 include/tty_lib2.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <unistd.h>
```

Functions

- void [config_tty](#) (const char *tty_port, struct termios *tty, unsigned int baud, int *serial_port)

4.12.1 Function Documentation

4.12.1.1 config_tty()

```
void config_tty (
    const char * tty_port,
    struct termios * tty,
    unsigned int baud,
    int * serial_port )
```

Definition at line 8 of file config_tty.cpp.

```
9 {
10
11     *serial_port = open(tty_port, O_RDWR);
12
13     // Check for errors
14     if (*serial_port < 0) {
15         printf("Error %i from open: %s\n", errno, strerror(errno));
16     }
17
18
19     // Create new termios struct, we call it 'tty' for convention
20     // No need for "= {0}" at the end as we'll immediately write the existing
21     // config to this struct
22     //struct termios tty;//no needed here as is received in function argument
23
24     // Read in existing settings, and handle any error
25     // NOTE: This is important! POSIX states that the struct passed to tcsetattr()
26     // must have been initialized with a call to tcgetattr() otherwise behaviour
27     // is undefined
28     if(tcgetattr(*serial_port, tty) != 0) {
29         printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
30     }
31
32     tty->c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
33     //tty->c_cflag |= PARENB; // Set parity bit, enabling parity
34
35     tty->c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)
36     //tty->c_cflag |= CSTOPB; // Set stop field, two stop bits used in communication
37
38
39     tty->c_cflag &= ~CSIZE; // Clear all the size bits, then use one of the statements below
40     //tty->c_cflag |= CS5; // 5 bits
41     //tty->c_cflag |= CS6; // 6 bits
42     //tty->c_cflag |= CS7; // 7 bits
43     tty->c_cflag |= CS8; // 8 bits (most common)
44
45
46     tty->c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
47     //tty->c_cflag |= CRTSCTS; // Enable RTS/CTS hardware flow control
48
49     tty->c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)
50
51     //In canonical mode, input is processed when a new line character is received.
52     tty->c_lflag &= ~ICANON; // non-canonical
53
54     //If this bit is set, sent characters will be echoed back.
55     tty->c_lflag &= ~ECHO; // Disable echo
56     tty->c_lflag &= ~ECHOE; // Disable erasure
57     tty->c_lflag &= ~ECHONL; // Disable new-line echo
58
59     tty->c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
60
61     tty->c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
62
63     tty->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of
64     // received bytes
65
66     tty->c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
67     tty->c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
68     // tty->c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT IN LINUX)
69     // tty->c_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT IN LINUX)
```

```

70
71 /*VMIN = 0, VTIME = 0: No blocking, return immediately with what is available
72 VMIN > 0, VTIME = 0: This will make read() always wait for bytes (exactly how many is determined by
   VMIN), so read() could block indefinitely.
73 VMIN = 0, VTIME > 0: This is a blocking read of any number of chars with a maximum timeout (given by
   VTIME). read() will block until either any amount of data is available, or the timeout occurs. This happens to
   be my favourite mode (and the one I use the most).
74 VMIN > 0, VTIME > 0: Block until either VMIN characters have been received, or VTIME after first
   character has elapsed. Note that the timeout for VTIME does not begin until the first character is received.
75 type of VMIN and VTIME: cc_t (1B)*/
76 tty->c_cc[VTIME] = 0;
77 tty->c_cc[VMIN] = 1; // wait one byte
78
79 //B0, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200,
   B38400, B57600, B115200, B230400, B460800
80 // Set in/out baud rate to be 9600
81 cfsetispeed(tty, baud);
82 cfsetospeed(tty, baud);
83 //cfsetspeed(tty, B9600); //set both input and output
84
85 //cfsetispeed(tty, 104560); //Specifying a custom baud rate when using GNU C
86 //cfsetospeed(tty, 104560);
87
88 /*Other option for custom baud rate*/
89 /*
90 // #include <termios.h> This must be removed!
91 // Otherwise we'll get "redefinition of struct termios" errors
92 #include <sys/ioctl.h> // Used for TCGETS2/TCSETS2, which is required for custom baud rates
93 struct termios2 tty;
94 // Read in the terminal settings using ioctl instead
95 // of tcsetattr (tcsetattr only works with termios, not termios2)
96 ioctl(fd, TCGETS2, tty);
97 // Set everything but baud rate as usual
98 // ...
99 // ...
100 // Set custom baud rate
101 tty->c_cflag &= ~CBAUD;
102 tty->c_cflag |= CBAUDEX;
103 // On the internet there is also talk of using the "BOTHER" macro here:
104 // tty->c_cflag |= BOTHER;
105 // I never had any luck with it, so omitting in favour of using
106 // CBAUDEX
107 tty->c_ispeed = 123456; // What a custom baud rate!
108 tty->c_ospeed = 123456;
109 // Write terminal settings to file descriptor
110 ioctl(*serial_port, TCSETS2, tty);
111 */
112
113 // Save tty settings, also checking for error
114 if (tcsetattr(*serial_port, TCSANOW, tty) != 0) {
115     printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
116 }
117 /*****/
118 /*WRITING*/
119 /*****/
120 //unsigned char msg[] = { 'H', 'e', 'l', 'l', 'o', '\r' };
121 //write(*serial_port, msg, sizeof(msg));
122
123 /*****/
124 /*READING*/
125 /*****/
126 // Allocate memory for read buffer, set size according to your needs
127 //char read_buf [256];
128
129 // Normally you wouldn't do this memset() call, but since we will just receive
130 // ASCII data for this example, we'll set everything to 0 so we can
131 // call printf() easily.
132 //memset(&read_buf, '\0', sizeof(read_buf));
133
134 // Read bytes. The behaviour of read() (e.g. does it block?,
135 // how long does it block for?) depends on the configuration
136 // settings above, specifically VMIN and VTIME
137 //int num_bytes = read(*serial_port, &read_buf, sizeof(read_buf));
138
139 // n is the number of bytes read. n may be 0 if no bytes were received, and can also be -1 to signal an
   error.
140 //if (num_bytes < 0) {
141 //    printf("Error reading: %s", strerror(errno));
142 //    return 1;
143 //}
144
145 // Here we assume we received ASCII data, but you might be sending raw bytes (in that case, don't try
   and
146 // print it to the screen like this!)
147 //printf("Read %i bytes. Received message: %s", num_bytes, read_buf);
148
149 //close(serial_port);

```



```

150
151 }

```

4.13 src/config_tty.cpp File Reference

```
#include "tty_lib2.h"
```

Functions

- void [config_tty](#) (const char *tty_port, struct termios *tty, unsigned int baud, int *serial_port)

4.13.1 Function Documentation

4.13.1.1 config_tty()

```

void config_tty (
    const char * tty_port,
    struct termios * tty,
    unsigned int baud,
    int * serial_port )

```

Definition at line 8 of file config_tty.cpp.

```

9 {
10
11     *serial_port = open(tty_port, O_RDWR);
12
13     // Check for errors
14     if (*serial_port < 0) {
15         printf("Error %i from open: %s\n", errno, strerror(errno));
16     }
17
18
19     // Create new termios struct, we call it 'tty' for convention
20     // No need for "= {0}" at the end as we'll immediately write the existing
21     // config to this struct
22     //struct termios tty;//no needed here as is received in function argument
23
24     // Read in existing settings, and handle any error
25     // NOTE: This is important! POSIX states that the struct passed to tcsetattr()
26     // must have been initialized with a call to tcgetattr() otherwise behaviour
27     // is undefined
28     if(tcgetattr(*serial_port, tty) != 0) {
29         printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
30     }
31
32     tty->c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
33     //tty->c_cflag |= PARENB; // Set parity bit, enabling parity
34
35     tty->c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)
36     //tty->c_cflag |= CSTOPB; // Set stop field, two stop bits used in communication
37
38
39     tty->c_cflag &= ~CSIZE; // Clear all the size bits, then use one of the statements below
40     //tty->c_cflag |= CS5; // 5 bits
41     //tty->c_cflag |= CS6; // 6 bits
42     //tty->c_cflag |= CS7; // 7 bits
43     tty->c_cflag |= CS8; // 8 bits (most common)
44
45

```

```

46     tty->c_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)
47     //tty->c_cflag |= CRTSCTS; // Enable RTS/CTS hardware flow control
48
49     tty->c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)
50
51     //In canonical mode, input is processed when a new line character is received.
52     tty->c_lflag &= ~ICANON; // non-canonical
53
54     //If this bit is set, sent characters will be echoed back.
55     tty->c_lflag &= ~ECHO; // Disable echo
56     tty->c_lflag &= ~ECHOE; // Disable erasure
57     tty->c_lflag &= ~ECHONL; // Disable new-line echo
58
59     tty->c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
60
61     tty->c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
62
63     tty->c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of
64     received bytes
65
66     tty->c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)
67     tty->c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed
68     // tty->c_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT IN LINUX)
69     // tty->c_oflag &= ~NOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT IN LINUX)
70
71     /*VMIN = 0, VTIME = 0: No blocking, return immediately with what is available
72     VMIN > 0, VTIME = 0: This will make read() always wait for bytes (exactly how many is determined by
73     VMIN), so read() could block indefinitely.
74     VMIN = 0, VTIME > 0: This is a blocking read of any number of chars with a maximum timeout (given by
75     VTIME). read() will block until either any amount of data is available, or the timeout occurs. This happens to
76     be my favourite mode (and the one I use the most).
77     VMIN > 0, VTIME > 0: Block until either VMIN characters have been received, or VTIME after first
78     character has elapsed. Note that the timeout for VTIME does not begin until the first character is received.
79     type of VMIN and VTIME: cc_t (1B)*/
80     tty->c_cc[VTIME] = 0;
81     tty->c_cc[VMIN] = 1; // wait one byte
82
83     //B0, B50, B75, B110, B134, B150, B200, B300, B600, B1200, B1800, B2400, B4800, B9600, B19200,
84     B38400, B57600, B115200, B230400, B460800
85     // Set in/out baud rate to be 9600
86     cfsetispeed(tty, baud);
87     cfsetospeed(tty, baud);
88     //cfsetspeed(tty, B9600); //set both input and output
89
90     //cfsetispeed(tty, 104560); //Specifying a custom baud rate when using GNU C
91     //cfsetospeed(tty, 104560);
92
93     /*Other option for custom baud rate*/
94     /*
95     // #include <termios.h> This must be removed!
96     // Otherwise we'll get "redefinition of struct termios" errors
97     #include <sys/ioctl.h> // Used for TCGETS2/TCSETS2, which is required for custom baud rates
98     struct termios2 tty;
99     // Read in the terminal settings using ioctl instead
100    // of tcsetattr (tcsetattr only works with termios, not termios2)
101    ioctl(fd, TCGETS2, tty);
102    // Set everything but baud rate as usual
103    // ...
104    // ...
105    // Set custom baud rate
106    tty->c_cflag &= ~CBAUD;
107    tty->c_cflag |= CBAUDEX;
108    // On the internet there is also talk of using the "BOTHER" macro here:
109    // tty->c_cflag |= BOTHER;
110    // I never had any luck with it, so omitting in favour of using
111    // CBAUDEX
112    tty->c_ispeed = 123456; // What a custom baud rate!
113    tty->c_ospeed = 123456;
114    // Write terminal settings to file descriptor
115    ioctl(*serial_port, TCSETS2, tty);
116    */
117
118    // Save tty settings, also checking for error
119    if (tcsetattr(*serial_port, TCSANOW, tty) != 0) {
120        printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
121    }
122    /******
123    /*WRITING*/
124    /******
125    //unsigned char msg[] = { 'H', 'e', 'l', 'l', 'o', '\r' };
126    //write(*serial_port, msg, sizeof(msg));
127
128    /******
129    /*READING*/
130    /******
131    // Allocate memory for read buffer, set size according to your needs

```

```

127 //char read_buf [256];
128
129 // Normally you wouldn't do this memset() call, but since we will just receive
130 // ASCII data for this example, we'll set everything to 0 so we can
131 // call printf() easily.
132 //memset(&read_buf, '\0', sizeof(read_buf));
133
134 // Read bytes. The behaviour of read() (e.g. does it block?,
135 // how long does it block for?) depends on the configuration
136 // settings above, specifically VMIN and VTIME
137 //int num_bytes = read(*serial_port, &read_buf, sizeof(read_buf));
138
139 // n is the number of bytes read. n may be 0 if no bytes were received, and can also be -1 to signal an
140 //error.
141 //if (num_bytes < 0) {
142 //    printf("Error reading: %s", strerror(errno));
143 //    return 1;
144 //}
145
146 // Here we assume we received ASCII data, but you might be sending raw bytes (in that case, don't try
147 //and
148 // print it to the screen like this!)
149 //printf("Read %i bytes. Received message: %s", num_bytes, read_buf);
150
151 //close(serial_port);
152 }

```

4.14 src/main.cpp File Reference

```
#include "lib_grupo2.h"
```

Functions

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

4.14.1 Function Documentation

4.14.1.1 main()

```

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

```

Definition at line 3 of file main.cpp.

```

4 {
5     int tiempo_preparacion;
6     int tiempo_lectura;
7     unsigned int read_buf;
8     int num_bytes;
9     int cont;
10    int cont2;
11    struct termios tty;
12    int serial_port;
13
14    int num_part;
15
16    //esto permite conectar el arduino a este programa solo se configura la salida
17    //arduino y la velocidad de transmision de datos
18    cout << "Ingrese la cantidad de participantes: "<< endl;

```

```

19     cin >> num_part;
20     participantes participante[num_part];
21     //write (serial_port, "S", sizeof(char));
22     //sleep(10);
23
24
25 //funcion para que el tiempo de preparacion y tiempo de lectura este muerto mientras transcurre
26     for (cont=0; cont < num_part; cont ++){
27         {
28             config_tty ("/dev/ttyS0",&tty,B9600,&serial_port);
29
30             participante[cont].set_participantes();
31             tiempo_preparacion=stoi(argv[1]); //prende el amarillo
32             tiempo_lectura=stoi(argv[2]); //prende el verde
33
34             sleep(2);
35             write (serial_port, "s", sizeof(char));
36
37             cout << ";PREPARETE!" << endl ;
38             sleep(tiempo_preparacion);
39             write (serial_port, "x", sizeof(char));
40             cout << ";EMPIEZA A PULSAR!" << endl;
41             sleep(tiempo_lectura);
42
43             if (num_bytes > 0) {
44                 cout << num_bytes << endl;
45             }
46
47             write (serial_port, "S", sizeof(char));
48             num_bytes = read(serial_port,&read_buf,sizeof(read_buf));
49             cout << "llego hasta aqui:" << read_buf << " en " << argv[2] << " segundos" << endl;
50             participante[cont].set_pushed((char) read_buf);
51
52             //cout << "su puntaje Bytes: " << read_buf << endl;
53             //cout << "Su puntaje es: " << num_bytes << endl;
54             printf ("\n %d\n ", (char) read_buf);
55             close (serial_port);
56
57         }
58     }
59     for (cont2 = 0; cont2 < num_part; cont2 ++){
60     {
61         //Impresion de los datos
62         participante[cont2].ImprimirInfo();
63     }
64     return 0;
65 }

```

4.15 src/participant.cpp File Reference

```
#include "lib_grupo2.h"
```