Arduboy

Starters Guide

Stijn Caerts

Jeugd, Cultuur en Wetenschap vzw

Jeugd, Cultuur en Wetenschap vzw
Youth, Culture and Science Foundation

stijn.caerts.be – www.jeugdcultuurenwetenschap.be

First edition, August 2019

Contents

Introduction to C++

1. Introduction

1.1 What is C++?

C++ is an object-oriented, general-purpose programming language. Programs for **Arduino** and the **Arduboy** are typically written in C++. You do not need to know and understand the entire programming language before you can get started with programming. That is why we provide a brief overview of the most important concepts that you need to get started.

The following chapters deal with variables, data types, control structures (if-then-else, for, while), functions and procedures. Finally, two additional chapters elaborate on the possibilities of C++, namely arrays and lists, and classes and objects.

For an interactive and more extensive introduction to C++, you can go to W3Schools (`https://www.w3schools.com/cpp/`).

1.2 Syntax

1.2.1 Semicolons

A semicolon `;` is placed after every instruction in C++. This semicolon tells the compiler that the instruction has ended. If you forget to place a semicolon, the program is not correct and will not compile. The compiler will show an error message.

■ **Example 1.1** Make note of the `;` at the end of each line.

```
int a = 42; // ; here
int b = 10; // ; another here
```

◂

1.2.2 Curly Braces

Curly braces {} (also called curly brackets) are used to indicate blocks of code. The braces must be balanced; for every opening brace { there must be a matching closing brace }. The use of braces is further illustrated later with control structures (if-then-else, for, ..), functions and classes.

■ **Example 1.2**

```
while(keepRunning)
{
  // this is a block of code
  // note the { to start this block
  // and the } to end it
}
```

◂

1.2.3 Comments

To increase the readability of your code, it is sometimes useful to add comments. In these comments you can describe what your code does and why. Comments can help remind you of important knowledge when viewing your code again in the future.

In C++ there are two different ways to add comments.
Comments on a single line are indicated by //. All text after the // (until the end of the line) is considered a comment and will not be executed.

■ **Example 1.3 — A Comment.**

```
// This line is a comment
int a = 42;
```

◂

Comments on multiple lines start with /* and end with */. All text between /* and */ is ignored by the compiler.

■ **Example 1.4 — Multi-line comments.**

```
/*
  This is a comment that is so very long
  that it really must span multiple lines.
*/
int a = 42;
```

◂

2. Variables and Types

Variables are used to keep track of information or data inside a program. A variable is a place in memory, with a self-chosen name, where data can be stored. In C++, each variable has a type. A type corresponds to the type of data that can be stored in such a variable.

2.1 Data Types

Based on the data type of a variable, memory is reserved for storing information. The following primitive data types are defined in C++:

Definition 2.1.1 — Primitive Data Types.

bool	Boolean value: `true` or `false`
char	Character
int	Integer
float	Floating-point (single-precision)
double	Floating-point (double-precision)

These types are used with variables, but also with functions. These primitive data types can also be used to compose more complex types, such as arrays and classes.

★ The type of a variable is important for the compiler because the type determines how much memory must be reserved for the variable. For example, 32 bits are required to store a `float` variable, while 64 bits are required to store a double-precision (`double`) decimal number.

2.2 Variables

A variable can be thought of as a storage location with a name is attached. A value can then be placed at this storage location.

2.2.1 Declaration

Before information can be stored in a variable, the variable must first be *defined* or *declared*. The declaration of a variable tells the compiler that space must be reserved in memory to store data of the given data type.

You can declare a new variable with an instruction of this form:

```
type name_of_variable;
```

■ **Example 2.1 — Declare an *integer* variable.**

```cpp
// Declare a new integer variable named a
int a;
```

◂

When defining a new variable you can also immediately assign a value to it.

■ **Example 2.2 — Declare an *integer* variable and assign it a value.**

```cpp
/*
  Declare a new variable of type int with name a,
  and assign the value 42
*/
int a = 42;
```

◂

When choosing a variable name you must take the following rules into account:

- names must begin with a letter or a hyphen (_);
- names may contain letters, numbers and low dashes;
- names are case-sensitive;
- spaces and special characters are not allowed;
- C++ keywords (such as `int`, `if`, ...) cannot be used as a name.

■ **Example 2.3 — Declaring and initializing different data types.**

```cpp
bool b = true;
char c = 'q';
int i = 42;
float f = 3.14;
double d = 3.14159265;
```

◂

2.2.2 Scope

The *scope* of a variable is the place in the program where you can use a defined variable. You can also consider this as the lifetime of the variable.

The scope of a variable is determined by where the variable was declared. We can divide this into 3 different types of places.

1. in a function or block: **local variable**,
2. in the definition of a function: **parameters**,
3. outside all functions: **global variable**.

In general the scope of a variable is limited to the block (indicated by braces {}) in which the variable is defined.

■ **Example 2.4 — Variable scope.**

```
// global variable g
int g = 12;

int double(int x) {
        // x is a parameter here
        return x * 2;
}
// x cannot be used here, outside of the double function

void setup() {
  // local variables a and b
  int a = 42;

  // global variables can be used anywhere after being declared
  int b = a + g;

  /*
    after declaration, the type of a variable does not have to
    be specified to use that variable
  */
  g = b;
}
// variables a and b cannot be used outside the setup() function
```

◂

2.2.3 Operators

To alter the value of variables, we can use *operators*. Below is an overview of the most important operators in C++.

Arithmetic Operators

Arithmetic operators are used to perform commonly used mathematical operations. In these examples we use the variables $A = 10$ and $B = 3$

Operator	Description	Example
+	Addition	A + B $\rightarrow 13$
-	Subtraction	A - B $\rightarrow 7$
*	Multiplication	A * B $\rightarrow 30$
/	Division	A / B $\rightarrow 3$
%	Modulus (remainder after division)	A % B $\rightarrow 1$
++	Increase the value by 1	A++ $\Rightarrow$ A $\rightarrow 11$
--	Decrease the value by 1	A-- $\Rightarrow$ A $\rightarrow 9$

Assigning Operators

The operator for assigning a value to a variable is the equal sign (=). In addition to this assignment operator, there are other variants, which are shorter notations for common arithmetic operations followed by an assignment.

Operator	Example	Equivalent
`=`	`A = 12`	`A = 12`
`+=`	`A += 2`	`A = A + 2`
`-=`	`A -= 3`	`A = A - 3`
`*=`	`A *= 4`	`A = A * 4`

Comparison Operators

Comparison operators are used to compare two values. The result of the comparison is a Boolean value, `true` or `false`.

Operator	Description	Example
`==`	Equal	`10 == 3` → `false`
`!=`	Not equal	`10 != 3` → `true`
`>`	Greater than	`10 > 10` → `false`
`<`	Less than	`5 < 10` → `true`
`>=`	Greater than or equal to	`10 >= 10` → `true`
`<=`	Less than or equal to	`5 <= 10` → `true`

Logical Operators

Logical operators are used to combine multiple Boolean values in a formula into one truth value.

Operator	Description	Example
`&&`	Logical and (AND)	`(true && false)` → `false`
`||`	Logical or (OR)	`(true || false)` → `true`
`!`	Logical negation (NOT)	`! true` → `false`

For more examples of using logical operators, you can take a look at truth tables [5].

3. Control Structures

Control structures determine the course of a program. For example, it is possible to execute pieces of code only in certain cases, or to repeat a block of code many times.

3.1 If-then-else

The simplest control structure is the *if* statement. This can be further expanded with an *else* block for further control over the execution order.

3.1.1 If-then

The if statement is used to execute code that is intended for specific cases. Only when the specified condition is met, will the instructions in the body of the if statement be executed. If the condition is not met, the instructions in the body will be skipped.

Definition 3.1.1 — If.

```
if (condition) {
  /*
    instructions between these braces are only
    performed if the condition evaluates to true
  */
}
```

3.1.2 Else

After an if statement, an else block can optionally be placed. The instructions in this block will only be executed if the condition of the if statement is **not** met. The code then looks like this.

Definition 3.1.2 — If-then-else.

```
if (condition) {
  /*
    instructions in this block are executed
    if the condition evaluates to true
  */
} else {
  /*
  instructions in this block are executed
  if the condition evaluates to false
  */
}
```

3.1.3 Nested if-statements

When there are several conditions that require testing, you can use a nested if statement. Place a new if statement inside the else block where you test the following condition. This looks like this.

■ **Example 3.1 — Nested if-statements.**

```
int x = 42;

if (x < 10) {
  x = x * 2;
} else {
  if (x > 50) {
    x = x - 25;
  } else {
    x++;
  }
}
```

◂

Because nested if statements quickly become unclear, this code can be simplified as follows.

■ **Example 3.2 — Else-if.**

```
int x = 42;

if (x < 10) {
  x = x * 2;
} else if (x > 50) {
  x = x - 25;
} else {
  x++;
}
```

◂

3.2 Switch

If you want to test an expression against multiple values, you can also use a *switch* statement instead of nested if statements.

Definition 3.2.1 — Switch.

```
switch (expression) {
  case x:
    // instructions
    break;
  case y:
    // instructions
    break;
  default:
    // instructions
}
```

Here the expression in the `switch` is evaluated once. Its value is then compared with the value in each `case`. If the values match, the corresponding block of instructions is executed.

The `break` statement ensures that, after a matching `case` has been found, matches with other `case`s should not be searched for. The `default` keyword can then be used to execute instructions when no matching `case` is found.

■ **Example 3.3 — Switch – days of the week.**

```
int day = 5;
char c;

switch (day) {
  case 1:
    c = 'M';
    break;
  case 2:
    c = 'T';
    break;
  case 3:
    c = 'W';
    break;
  case 4:
    c = 'T';
    break;
  case 5:
    c = 'F';
    break;
  case 6: // same as case 7
  case 7:
    c = 'S';
    break;
  default:
    c = 'X';
}
```

Exercise 3.1 — Switch – days of the week. After executing the above code (Example 3.3), what character is stored in variable `c`?

3.3 While Loop

If you want to execute a piece of code multiple times in a row, you can use a loop. The `while` loop contains a condition and a body with instructions. As long as the condition is true, the instructions in the body are repeatedly executed.

Definition 3.3.1 — While.

```
while (condition) {
  // body of the loop

  // instructions
}
```

Example 3.4 — While. This code calculates the smallest multiple `vv` of the number `x` that is greater than 24.

```
int x = 3;
int vv = 0;

while (vv <= 24) {
  vv += x;
}
```

3.4 For Loop

Often loops occur where the number of repetitions is known in advance. These loops then have the following shape.

Example 3.5 A loop with only 10 repetitions.

```
int i = 0;

while (i < 10) {
  // instructions

  i++;
}
```

Before the start of the loop, the variable `i` is declared, then used inside the loop to count the number of iterations (repetitions). The value of `i` is incremented at the end of each iteration (`i++`). This type of loop can be written more easily as a `for` loop.

Definition 3.4.1 — For.

```
for (initialization; condition; increment/decrement) {
  // body, contains a block of code that is repeated each time
}
```

At which:

initialization is executed once before the start of the loop. Often this is the declaration of a *loop variable*, for example `int i = 0`.

condition represents the condition of the loop.

increment/decrement is executed every time after iteration of the loop. Often this is a manipulation of the loop variable, such as `i++`.

The code from Example 3.5 can therefore be written using a `for` loop as follows.

■ **Example 3.6**

```
for (int i = 0; i < 10; i++) {
  // instructions
}
```

◀

4. Functions and Procedures

Functions and procedures are blocks of code that you can reuse in various places throughout your code. Instead of writing the same code over and over, it is better to place the common code in a function. This also increases the readability of the code.

4.1 Function Definition

The definition of a function is structured as described below.

Definition 4.1.1 — Function.

```
type function_name(arguments) {
  // here comes the common code

  return result;
}
```

The definition of a function consists of the heading and the body. The type of function is first noted in the header. This is followed by the name of the function, and finally the arguments of the function are placed in brackets. This is followed by the body of the function within the braces.

4.1.1 Type

The type of the function is the type of value that the function returns. A function is called from the code, after which the code is executed in the body of the function. The result of these calculations is then returned to the place where this function call took place. The result can then be used there.

The primitive data types from Definition 2.1.1 can be used as a type of a function. It is also possible that a function returns no result. We call such a function a procedure. The *return type* for procedures is `void`, the empty type.

4.1.2 Arguments

Information can be passed into the function via the *arguments* or *parameters*. These parameters behave as variables within the body of the function. Multiple parameters are separated from each other in the header using a comma.

■ Example 4.1 — Function – addition.

```
int addemup(int a, int b, int c) {
  int sum = a + b + c;
  return sum;
}
```

◀

■ Example 4.2 — Function – minimum.

```
int minimum(int a, int b) {
  if (a < b) {
    return a;
  } else {
    return b;
  }
}
```

◀

4.2 Calling Functions

Once a function has been defined, this function can be called from other places in the code. Call a function as follows:

```
function_name(argument);
```

The function `minimum()` from Example 4.2 can therefore be called as follows.

■ Example 4.3 — Calling the function *minimum()*.

```
int x = 42;
int y = 13;

int result = minimum(x, y);
// result will equal 13
```

◀

II Arduboy

5. Arduino

5.1 Program Structure

Every Arduino program has the same structure. The most important features are the global variables, the `setup()` procedure and the `loop()` procedure. We discuss these in more detail in the following sections.

Definition 5.1.1 — Arduino Program Structure.

```
// Declaration of global variables

void setup() {

}

void loop() {

}
```

5.1.1 Global Variables

Any global variables are declared at the top of the Arduino program. These global variables can then be used anywhere in the program, both in the `setup()` and `loop()` function.

5.1.2 The `setup()` procedure

The `setup()` procedure is executed once when the program starts.

★ In the `setup()` procedure you are probably going to initialize the values of various variables. It is important that you declare these variables **outside** the `setup()` function. Otherwise these variables are local and cannot be used outside the scope of the `setup()` function.

5.1.3 The `loop()` procedure

The `loop()` procedure does exactly what the name suggests. The code in this procedure is executed over and over again. The actual code of the Arduino program is written here. This repetition continues until the Arduino is powered off or the reset button is pressed.

5.2 Using Libraries

When programming for Arduino, it is very common to use *libraries*. Libraries contain functions written by other developers. They aim to reduce the complexity of your code by implementing operations at a lower level. This means that you do not have to program all communication with a display manually, for example, but use the functionality of the library for this.

You use a library in your code in the following way:

```
#include <Library.h>
```

5.3 Arduino IDE

The Arduino IDE is an easy way to get started writing Arduino programs. This program is an *Integrated Development Environment*, which means that it contains all the tools to write, compile and upload programs to an Arduino. The Arduino IDE can be downloaded for free from the Arduino website [1].

[1] https://www.arduino.cc/en/Main/Software

6. Arduboy

6.1 Preparation

Before we can get started with programming games for the Arduboy, we first have to adjust some settings in the Arduino IDE. For a manual with screenshots you can go to the Arduboy Community [2].

6.1.1 Installing the Arduboy2 library

The *Arduboy2* library contains a handy interface that ensures that we can easily access all functions of the device. This way we can fully focus on designing and programming the game itself.

Before we can use this library, it must first be installed in the IDE. You do this as follows:

1. Click **Tools** > **Manage Libraries...**
2. In the *Library Manager* window search for the library **Arduboy2**
3. Choose the latest version then click **Install**

This way easily install other libraries using this same process if you wish.

6.1.2 Installing the Arduboy as an Arduino board

1. Go to **File** > **Preferences**
2. In the *Preferences* screen, enter the following URL at **Additional Boards Manager URLs**: `https://arduboy.github.io/board-support/package_arduboy_index.json`
3. Click OK to close the *Preferences* window
4. Go to **Tools** > **Board** > **Boards Manager...**
5. Zoek naar **Arduboy** en klik op **Installeren**

6.1.3 Board Selection

To let the IDE know that you are programming for the Arduboy, you must select the board in the IDE.

1. Go to **Tools** > **Board**
2. Choose **Arduboy** from the list.

6.2 Program Structure

```
#include <Arduboy2.h>

Arduboy2 arduboy;

void setup() {
  arduboy.begin();
}

void loop() {
  if (arduboy.nextFrame()) {
    arduboy.clear();

    // GAME LOGIC

    arduboy.display();
```

```
    }
}
```

The code fragment above gives a template from which you can start to program an Arduboy game. The Arduboy2 library is included on the first line. A `Arduboy2` object is then created. This object is used to invoke the functionality of the library. Finally, during `setup()`, the functionality of the Arduboy is initialized.

6.3 The Arduboy2 Library

Here we provide an overview of the most important functions in the Arduboy2 library [1]. These functions can be used in this way:

`arduboy.function_name();`

Optional parameters for the functions are *italic*.

6.3.1 Startup

These functions are used to properly initialize the Arduboy.

Library — begin(). Initialize the hardware, display the logo, etc. This function must be called once in the `setup()` procedure.

Library — initRandomSeed(). Choose a random value as *seed* for the random number generator. This method is most effective when it is called after a (semi-) random time.

6.3.2 Display

These functions relate to the screen of the Arduboy.

Library — DISPLAY. There are the following handy constants for the width and height of the screen. In addition, there are also constants for the colors of the screen.

- `WIDTH`
- `HEIGHT`
- `BLACK`
- `WHITE`

Library — clear(). Clear the display buffer. The entire screen is set to black.

Library — display(). Paint the contents of the display buffer to the screen.

Library — setFrameRate(fps). Sets the desired frame rate.

Parameter
- `int fps`: desired frame-rate in frames per second

Library — nextFrame(). Indicates when it is time to display the next frame.

Returns

- `bool`: `true` when it's time for the next frame, otherwise `false`

Drawing

These functions are used to draw shapes on the screen of the Arduboy.

Library — drawPixel(x, y, *color=WHITE*). Draws a single pixel in the specified color.

Parameters
- `int x`: x coordinate of the pixel
- `int y`: y coordinate of the pixel
- `int color` (optional): color → `WHITE` or `BLACK`

Library — drawCircle(x, y, r, *color=WHITE*). Draws a circle.

Parameters
- `int x`: x coordinate of the center of the circle
- `int y`: y coordinate of the center of the circle
- `int r`: radius of the circle
- `int color` (optional): color → `WHITE` of `BLACK`

Library — drawRect(x, y, w, h, *color=WHITE*). Draws a rectangle.

Parameters
- `int x`: x coordinate of the top left corner
- `int y`: y coordinate of the top left corner
- `int w`: width
- `int h`: height
- `int color` (optional): color → `WHITE` of `BLACK`

Library — drawRoundRect(x, y, w, h, r, *color=WHITE*). Draws a rectangle with rounded corners.

Parameters
- `int x`: x coordinate of the top left corner
- `int y`: y coordinate of the top left corner
- `int w`: width
- `int h`: height
- `int r`: radius of the rounding
- `int color` (optional): color → `WHITE` of `BLACK`

Library — drawTriangle(x0, y0, x1, y1, x2, y2, *color=WHITE*). Draws a triangle.

Parameters
- `int x0, int x1, int x2`: x coordinates of the angles
- `int y0, int y1, int y2`: y coordinates of the angles
- `int color` (optional): color → `WHITE` of `BLACK`

Library — drawLine(x0, y0, x1, y1, *color=WHITE*). Draws a line between the specified points.

Parameters

- `int x0, int x1`: x coordinates of the points
- `int y0, int y1`: y coordinates of the points
- `int color` (optional): color → `WHITE` of `BLACK`

Library — drawFastVLine(x, y, h, *color=WHITE*). Draws a vertical line.

Parameters

- `int x`: x coordinate of the upper end
- `int y`: y coordinate of the upper end
- `int h`: height
- `int color` (optional): color → `WHITE` of `BLACK`

Library — drawFastHLine(x, y, w, *color=WHITE*). Draws a horizontal line.

Parameters

- `int x`: x coordinate of the left end
- `int y`: y coordinate of the left end
- `int w`: width
- `int color` (optional): color → `WHITE` of `BLACK`

Library — fillCircle(x, y, r, *color=WHITE*). Draws a filled circle.

Parameters

- `int x`: x coordinate of the center of the circle
- `int y`: y coordinate of the center of the circle
- `int r`: radius of the circle
- `int color` (optional): color → `WHITE` of `BLACK`

Library — fillRect(x, y, w, h, *color=WHITE*). Draws a filled rectangle.

Parameters

- `int x`: x coordinate of the top left corner
- `int y`: y coordinate of the top left corner
- `int w`: width
- `int h`: height
- `int color` (optional): color → `WHITE` of `BLACK`

Library — fillRoundRect(x, y, w, h, r, *color=WHITE*). Draws a filled rectangle with rounded corners.

Parameters
- `int x`: x coordinate of the top left corner
- `int y`: y coordinate of the top left corner
- `int w`: width
- `int h`: height
- `int r`: radius of the rounding
- `int color` (optional): color → `WHITE` of `BLACK`

Library — fillTriangle(x0, y0, x1, y1, x2, y2, *color=WHITE*). Draws a filled triangle.

Parameters
- `int x0, int x1, int x2`: x coordinates of the angles
- `int y0, int y1, int y2`: y coordinates of the angles
- `int color` (optional): color → `WHITE` of `BLACK`

Library — fillScreen(*color=WHITE*). Fills the screen with the chosen color.

Parameters
- `int color` (optional): color → `WHITE` of `BLACK`

Text
These functions are used to display text on the screen of the Arduboy.

Library — setCursor(x, y). Sets the location of the text cursor.

Parameters
- `int x`: x coordinate
- `int y`: y coordinate

Library — setTextSize(s). Sets the size of the text drawn.

Parameter
- `int s`: text size (≥ 1)

Library — setTextColor(c). Sets the text color.

Parameter
- `int c`: color → `WHITE` of `BLACK`

Library — setTextBackground(bg). Sets the text background color.

Parameter
- `int bg`: background color → `WHITE` of `BLACK`

Library — setTextWrap(w). Enables or disables text wrapping. If text wrapping is enabled, the text will be placed on a new line when it is too long to fit on the current line.

Parameter
- `bool w`: `true` to enable text wrap, `false` to disable

Library — print(content). Prints the specified content at the cursor location.

Parameter
- `content`: the content to be displayed

6.3.3 Buttons

With these functions your program can interact with the buttons of the Arduboy.

Library — BUTTONS. The following constants are defined for the buttons on the Arduboy.

- `A_BUTTON`
- `B_BUTTON`
- `LEFT_BUTTON`
- `RIGHT_BUTTON`
- `UP_BUTTON`
- `DOWN_BUTTON`

Buttons can easily be merged into a mask: `LEFT_BUTTON + A_BUTTON`

Library — pressed(buttons). Tests whether the specified buttons are pressed.

Parameters
- `int buttons`: bit mask of which buttons to test

Returns
- `bool`: `true` if all buttons in the specified mask are currently pressed, otherwise `false`

Library — notPressed(buttons). Tests if the specified buttons **not** are pressed.

Parameters
- `int buttons`: bit mask of which buttons to test

Returns
- `bool`: `true` if all buttons in the specified mask are currently **not** pressed, otherwise `false`

6.3.4 Physics

These functions are used to calculate interactions between different objects in your game.

Library — collide(rect1, rect2). Checks if two rectangles intersect.

Parameters

- `Rect rect1, Rect rect2`: rectangles → Rect(int x, int y, int width, int height)

Returns

- `bool`: `true` if the rectangles overlap

Library — collide(point, rect). Checks if a point is within a rectangle.

Parameters

- `Point point`: point → Point(int x, int y)
- `Rect rect`: rectangle → Rect(int x, int y, int width, int height)

6.3.5 LED

You can control the RGB LED of the Arduboy via these functions.

Library — setRGBled(red, green, blue). Set the brightness of the different colors in the RGB LED. The RGB LED are actually small individual red, green and blue LEDs that are placed very close together. By adjusting the brightness of the 3 colors, you can show different colors. The brightness of each LED can take a value from 0 (fully off) to 255 (fully on).

Parameters

- `int red`: red component
- `int green`: green component
- `int blue`: blue component

Library — setRGBled(color, val). Set the brightness of one of the colors without changing the others.

Parameters

- `int color`: color to change: `RED_LED`, `GREEN_LED` of `BLUE_LED`
- `int val`: brightness, value between 0 and 255

6.4 Emulator

If you are programming a game, you naturally want to be able to test whether it works the way you want. That is why there are emulators for the Arduboy.

6.4.1 ProjectABE

ProjectABE (`https://felipemanga.github.io/ProjectABE/`) is an online emulator for the Arduboy. You can test your own game by going to the start screen under **My Projects** > **New Game**. In the next screen you see the emulator, as in Figure 6.1.

In the Arduino IDE, click on **Sketch** > **Export compiled Binary file**. This will create a file with extension `.hex` in the same folder as your code. Then drag this file with extension `.hex` to the emulator. The emulator will immediately execute your code.
You can operate the emulator with the keyboard or by clicking on the buttons on the screen.

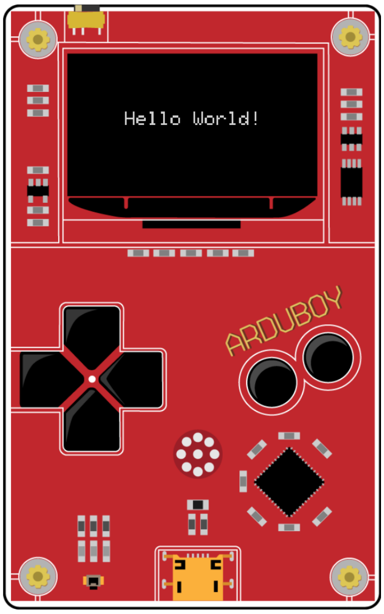

Figure 6.1: Project ABE

6.5 Upload to your Arduboy

Before placing the program on the Arduboy, make sure that the correct board is selected (see Section **??**). Then turn on the Arduboy and connect it to the computer via a USB cable. Select the correct port via **Tools** > **Port** and then choose the option that shows Arduino.
Now you are completely ready to place the program on the Arduboy. You just have to click on the arrow (**Upload**) and wait a while. If everything goes well, your program will soon be on the Arduboy.

Have fun!

Bibliography

[1] Arduboy2. *Library*. https://mlxxxp.github.io/documents/Arduino/libraries/Arduboy2/Doxygen/html/index.html (cited on page 27).

[2] Jonathan Holmes (crait) - Arduboy Community. *Make Your Own Arduboy Game: Part 1 - Setting Up Your Computer*. https://community.arduboy.com/t/make-your-own-arduboy-game-part-1-setting-up-your-computer/. June 2019 (cited on page 26).

[3] Tutorials Point. *C++ Tutorial*. https://www.tutorialspoint.com/cplusplus/.

[4] W3Schools. *C++ Tutorial*. https://www.w3schools.com/cpp/.

[5] Wikipedia. *Truth table*. https://en.wikipedia.org/wiki/Truth_table (cited on page 12).

Index

T

V