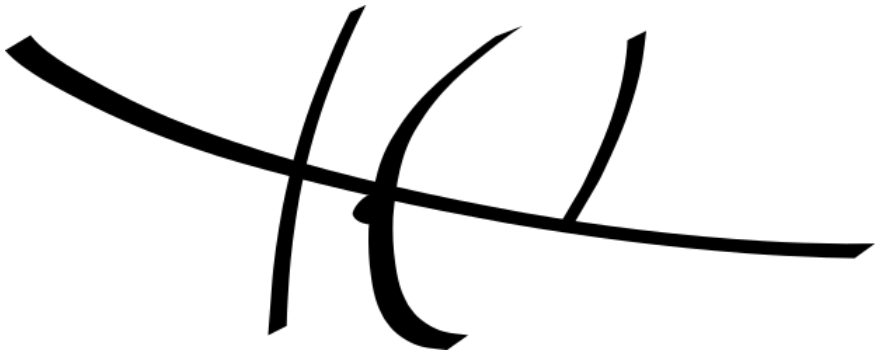
HCL Compiled Language



Design and implementation of a programming language

Aalborg University

Group SW407F18

Title:

Software 4: "HCL Compiled Language"

Group SW407F18

9. Februar - 2018

Semester:
4
Project theme:
Developing Applications - from users to Data, Algorithms and Tests - and back again
Project duration:
01/02/2018 - 21/12/2018
ECTS:
15
Supervisor:
Ingo Vanduijn

AALBORG UNIVERSITY
STUDENT REPORT

Abstract:
INSERT HERE

Authors:

Casper Weiss Bang

Jonas Hansen

Nichlas Ørts Lisby

Nicolaj Casanova Abildgaard

Thomas Højriis Knudsen

Tobias Lambek Jacobsen

Number of pages: 22.

Preface

This report is written during the fourth semester of the bachelor's degree in Software at Aalborg University, for the study board of computer science at the School of Information and Communication Technology. The report is written with the guidance of one supervisor.

The topic of the semester is 'Design, Definition and Implementation of Programming Languages'.

The system to be developed was chosen by the project group.

Terms and abbreviations used in the report:

WW : Whole Word

Contents

1 Introduction **3**
1.1 Preliminary interviews with the client 4

2 Analysis **5**
2.1 Languages for beginners 5
2.1.1 Scratch 5
2.1.2 Game maker 5
2.1.3 Python 6
2.1.4 Takeaways 6
2.2 Educational single-board computers 7
2.2.1 Raspberry Pi 7
2.2.2 Lego Mindstorm 7
2.2.3 Takeaway 7
2.3 Existing languages 8

3 Theory **14**

4 Implementation **15**

5 Conclusion **16**
5.1 Discussion 16
5.2 Conclusion 16
5.3 Future work 16

Bibliography **17**

I Appendix **19**

1 Introduction

In recent years, there has been a bigger focus on computer science, technology and programming in the danish school system. This is in large part due to the prediction that Denmark will lack skilled programmers, cyber security specialists and other skilled computer science workers in the near future. According to business insider, there will be a shortage of 19,000 IT professionals in Denmark by the year 2030.[1]

As a result, more students are being introduced to programming and programming languages during their duration at the danish gymnasiums. The goal of this introduction is to give the students a basic idea of how the computers they use everyday function, but also to impart an interest or motivation for studying programming in their continued education.

As programming languages are ever evolving and new languages are constantly designed it becomes relevant that students, who might be interested in studying programming, are introduced to the field in a way that teaches them the fundamentals of programming.

One of the most common tools for introducing programming and electrical engineering in danish gymnasiums, is the Arduino development board. The Arduino platform consists of its own dedicated IDE and language. The Arduino programming language is based on C/C++ and designed to both be easy to use, as well as being more hardware-oriented.[2] The Arduino platform maintains its simplicity by oftentimes shielding away a lot of complexity by ways that are non-present in most programming languages. Although this is an advantage in terms of simplicity, it might halter, or even frustrate, the student when learning other programming languages.

As such the motivation for this project is to create a new language for the Arduino platform, with a focus on promoting simplicity and the thought process that goes into programming, rather than difficult syntax and extraneous setup code.

From this the following initial problem statement is drafted:

How can a programming language for Arduino be optimized for learning without loosing the functionality of high-level languages

1.1 Preliminary interviews with the client

Analysis 2

2.1 Languages for beginners

There are a wide variety of different tools, games and languages with the intend to teach students how to program. These span from very specific game creation tools, to general purpose programming languages. In order to gather some inspiration for the HCL programming language some of the existing solutions, and most common languages for beginners have been analysed.

2.1.1 Scratch

Scratch is a programming language that lets the user program interactive stories, animations and games. It is one of the most common programming languages among youngsters and students who are just getting into programming. It is designed for children from the age of 8 to 16, and aims to help children "think creatively, reason systematically, and work collaboratively". In scratch the user do not actually have to write any code, since it utilizes drag and drop elements to build and structure your program. Although no exact programming is actually done, scratch helps to bring an understanding of the common programming concepts like variables, loops and conditionals.[3]

Scratch is one of the most popular block building programming languages, but there are a lot of different languages challenging it, for instance google's refinement of scratch which is called blockly.[4]

2.1.2 Game maker

Game maker is a powerful tool for developing games. It is arguably more complicated than scratch, but also more powerful and still aiming to be easy to use. Quoting their website:

"Making games development accessible to everyone means taking away the barriers to getting started. Using our intuitive 'Drag and Drop' development environment you can have your game up and running in a matter of minutes without ever having to write any code!".[5]

Besides using drag and drop like scratch, game maker also created a very simple programming language called "GML". GML resembles the syntax of common

languages like C and Java. However, it is a simple language, for instance type declaration is not needed or even supported. However, one thing to note is that the type system is indeed static, but types are inferred. As such types may not be altered at runtime.[6]

2.1.3 Python

Although not directly aimed at students or youngsters, python is a common entry point for many newcomers. Python also advertises it self as easy to learn for both new and experienced programmers. It provides a simple syntax, along with dynamic typing and type inference.[7] An example of pythons simple syntax free of all the usual ceremonials is easily represented just from the simple "Hello World" program.

Snippet 2.1: Hello World in python

```
1 print("Hello World")
```

now compare this to that of C++:

Snippet 2.2: Hello World in C++

```
1 #include <iostream>

3 using namespace std;

5 int main() {
6   cout << "Hello World" << endl;
7   return 0;
8 }
```

The C++ "Hello World" could potentially lead to a lot of questions, and all this just to be able to print to the screen.

Besides the easy and simple syntax python also includes a large collection of libraries to ease the development of different tasks.[8] This means you will be using less time reinventing the wheel and thus faster accomplishing the desired end goal.[9]

2.1.4 Takeaways

Although the above three languages only makes up a small subset of the analysed languages, they seem to be very representative for the general takeaways. Languages intended for beginners will often require little to no code to be written. This seems to be because they want to learn the students about the general programming concepts, before having to worry about syntax and formalities. Expanding on this, these languages also removes explicit type declaration, making the programmer write less ceremonial code.

Moreover the languages strives to have a simple, easy to learn syntax. Finally the languages generally want to provide a lot of methods and libraries for the user to use, so that he will not have to code these from scratch. This enables the programmer to spend more time telling what should be done, rather than how it should be done.

2.2 Educational single-board computers

While the group pretty quickly decided on developing for the Arduino platform, both because it is often used in the danish school system, as well as being the offered platform from AAU, it was still deemed necessary to look at other educational computer platforms.

Apart from the Arduino board, a lot of other single-board computers(SBCs) have been created with educational purposes in mind. To figure out why these SBCs are so popular for technical learning, some of the most popular ones were analysed. This should help to figure out which functionality the HCL programming language should include in order to comply with the educational intend of the SBCs.[10]

2.2.1 Raspberry Pi

The Raspberry Pi is one of the most widely known SBCs. It is a complete arm-based computer, and may run anything arm based on it, including both linux and windows based arm operating systems. This first of all makes it a common entry point into the GNU/Linux operating system stack, which is generally considered to be highly educational in regards to how computers work. The default Raspberry Pi distribution ships with both scratch and python, allowing the user to quickly get started with programming.[11]

On top of this, one of the major selling points of the Raspberry Pi is the GPIO[1] pins. These allow the user to attach all kind of devices to the board, this includes both sensors, lights, cameras and breadboards. The Raspberry Pi allows the user to easily interact with the GPIO pins, making it an ideal device for both IoT, home automation and robotics.[11]

2.2.2 Lego Mindstorm

Lego Mindstorm is a robot kit from Lego. The intend is to built a robot out of Lego, with the belonging motors and sensors which can then be controlled by the "head". The head is a small computer, which can control and monitor all the attached motors and sensors. The computer can be programmed from either a tablet or a computer using a drag and drop programming environment similar to that of scratches (2.1.1). [12]

[1] general purpose input output

2.2.3 Takeaway

Apart from being cheap, the big motivation behind these SBCs seem to be the extensibility through attachments. This allows the users to create something very concrete. For instance, it may be more rewarding for the user to get an actual lamp to blink or a car to drive, rather than just outputting something on the computer screen. The attachments will in the end allow the user to build something robotic, which is a common desire or goal for both new and experienced programmers.[13]

Since the Arduino also has the ability to connect various devices through its GPIO pins, it would be ideal to incorporate some easy to use GPIO handling into the HCL programming language.

2.3 Existing languages

Before doing any work on the groups own language,it was decided to look into the high-level languages that already exist for the Arduino platform, and what distinguishes them from each other. This was done in part to find out what kind of competition there may be for HCL, and in part to gain inspiration from the existing languages. There is a variety of languages to choose from when programming for the Arduino. Some of these are:

- Arduino C++ (the official Arduino language)
- Occam-pi
- NanoVM
- Juniper
- ArduinoBlocks

Arduino C++ is a subset of the C++ language with its own libraries, designed specifically for the Arduino. Because of this, it is compatible with the C and C++ languages. There are a few changes made to the language, but the basic functionality and syntax of C and C++ are still present, meaning users are able to use both the imperative- and the object-oriented paradigms of C and C++ respectively to program the Arduino.

The following is a brief overview of the basic syntax of the Arduino C++. Snippets 2.3 and 2.4 show an example program showcasing the syntax. All information is taken from the official Arduino reference[14].

The language requires two functions to be declared; the `setup` function which is called at startup, and the `loop` function with is called in an infinite loop after the `setup` code is run.

All *statements* are finished with a semi-colon.

Variables are declared by specifying a type and following it up with an identifier (snippet 2.3, line 11). Declarations can be prefixed with the `const` keyword to make

the variable immutable, or `static` to mark it as persistent between function calls. All variables have to be initialized before they are used.

Functions are declared with a return type, a name, parameters enclosed in parentheses, and a body encapsulated with curly-brackets (snippet 2.3, line 29). If the function does not return a value, the return type is set to `void`. If it is a class function, the class must be specified in the declaration (snippet 2.3, line 22).

Classes are declared with the keyword `class` followed by a name and the class body encapsulated in curly-brackets (snippet 2.4, line 1). The class' field is declared in the body, but isn't initialized here. Similarly, functions aren't given bodies in the class declaration either. Instead, the functions must be given a body outside of the class, and the variables must be initialized in the constructor function, also outside of the body.

Preprocessor directives such as including a header file, or setting up macros are written with a pound symbol (snippet 2.3, lines 6-9).

Comments can be written in two different ways. The end-of-line comment is prefixed with two slashes (snippet 2.3, line 6). The multi-line comment is prefixed with a slash followed by an asterisk (`"/*"`), and postfixed with an asterisk followed by a slash (`"*/"`) (snippet 2.3, line 1-4).

Snippet 2.3: An example program written in the Arduino language.

```
1  /*
2   * This program makes an LED connected to pin 13 on the arduino
3   * board blink on and off every 500 milliseconds
4   */
5
6  #include "Arduino.h" //The Arduino standard library
7  #include "Blinker.h" //The Blinker custom library
8
9  #define LED_PIN 13 //Symbolic constant
10
11 const int BLINK_DELAY = 250; //Constant, global integer variable
12
13 Blinker blinker(LED_PIN, BLINK_DELAY); //Object instantiation
14
15 void setup() {} //No setup code for this program
16
17 void loop()
18 {
19     blinker.Blink(); //Program blinks infinitely
20 }
21
22 Blinker::Blinker(int pin, int msDelay)
23 {
24   pinMode(pin, OUTPUT);
25   _pin = pin;
26   _msDelay = msDelay;
27 }
28
29 void Blinker::Blink()
30 {
```

```
31   digitalWrite(_pin, HIGH);
32   delay(_msDelay);
33   digitalWrite(_pin, LOW);
34   delay(_msDelay);
35 }
```

Snippet 2.4: The Blinker.h header file.

```
1  class Blinker
2  {
3    public:
4      Blinker(int pin, int msDelay); //Constructor
5      void Blink();
6    private:
7      int _pin;
8      int _msDelay;
9  };
```

Occam-pi is a language that focuses on making parallel programming easy for programmers. It is based on Occam that first appeared in 1983. The full documentation of Occam-pi can be read on the official website of the language, `concurrency.cc`, but following here is a short description of some of the syntax, and an example program in Snippet 2.5.

Variables are declared by initializing them with the colon-equals operator (lines 6-7). Type is automatically inferred by the expression on the right hand side of the initializing assignment.

Functions in Occam-pi are declared using the keyword `PROC` followed by a name and the function parameters enclosed in parentheses (line 5). Unlike C, Occam-pi doesn't use curly-brackets to encapsulate the body. Instead, the body is indented using a tab, similar to the Python language. The end of the function is marked using a colon. One of the key features of Occam-pi is the fact that it separates expressions that are evaluated sequentially and ones that are evaluated concurrently using the `SEQ` and `PAR` keywords.

Preprocessor directives are, like in C, prefixed with a pound symbol (line 3).

Comments are prefixed with two dashes (`"--"`) (line 1).

Snippet 2.5: An example program written in Occam-pi.

```
1  --This program makes two
2
3  #INCLUDE "plumbing.module" --Include module preprocessor directive
4
5  PROC main ()
6    SEQ --Expressions are evaluated sequentially
7      x := 4 + 3
8      y := x * 5
9    PAR --Expressions are evaluated concurrently
10     blink(12, 1000) --A function from the plumbing module
```

```
11      blink(13, 1000)
12  :
```

NanoVM is a virtual machine (VM) for the Atmel AVR ATMega8 CPU. It allows programmers to write code in a subset of Java that can be run by Arduino and similar microcomputers. The VM includes many of the features that have made the Java VM (JVM) popular such as object-oriented programming, automatic dynamic memory allocation, and garbage collection. It comes packaged with native classes such as Object, System, and PrintStream.

The following is a short description of the basic Java syntax. Snippets 2.6 and 2.7 showcase the syntax.

The program is started by calling the `main` function (snippet 2.6, line 5).

All statements are finished with a semi-colon.

In Java all variables and functions must be declared inside a class, and each class must be declared in a file for itself.

Class fields and methods are declared identically to how variables are declared in Arduino C++, except each field variable is prefixed with an optional access modifier (snippet 2.6, lines 3, 5). Variables can be made into class variables (called so because the variables belong to the class instead of instances of the class) with the `static` keyword before the name in the declaration. The same can be done to methods.

Comments are written exactly like in C and C++.

Snippet 2.6: An example program written in Java.

```
1  public class HelloWorld {
2
3    private static Printer printer = new printer(); //Class variable
4
5    public static void main(String[] args) { //Program start
6      printer.print("Hello World!");
7    }
8
9  }
```

Snippet 2.7: Printer class with one method written in Java.

```
1  public class Printer {
2    public void print(String str) { //Instance method
3      System.out.println(str);
4    }
5  }
```

The downsides of using NanoVM instead of Arduino C++ is, as stated on the

official website of NanoVM[2] that the VM's interpreter doesn't perform as well as C compiled to AVR code, and it reserves some of the RAM used for applications to run the VM. NanoVM also has to be installed on the CPU's flash memory, which will overwrite the bootloader.

Juniper is a functional programming language designed specifically for the Arduino. The intention behind creating Juniper was to provide a higher level language than Arduino C++, which at the same time was more suited for programming electronic devices[15].

The following is a description of the basic syntax of Juniper. Figure 2.8 showcases the syntax.

Variables are declared using the `let` keyword followed by a name, an equal sign, and an expression (line 4). Variables are by default immutable, ie. they can't be assigned another value after declaration, but they can be made mutable by inserting the `mutable` keyword before the name in their declaration.

Functions are declared using the `fun` keyword followed by a name, parameters enclosed in parentheses, an equal sign, and the function body encapsulated in parentheses (line 9).

Modules have the same purpose and functionality as they do in Occam-pi - to provide predefined functions to the programmer.

Comments in Juniper can be written in two different ways. End-of-line comments are prefixed with two slashes (just like C++ and Java). Multi-line comments are prefixed with a start parenthesis followed by an asterisk (`"(*"`), and post fixed with an asterisk followed by an end parenthesis (`"*)"`).

Snippet 2.8: Example program taken from the tutorial on Juniper's website.

```
1  module Blink
2  open(Prelude, Io, Time)
3
4  let boardLed = 13
5
6  let tState = Time:state()
7  let ledState = ref low()
8
9  fun blink() = (
10   let timerSig = Time:every(1000, tState);
11   let ledSig =
12     Signal:foldP(
13       fn (currentTimer, lastState) ->
14         Io:toggle(lastState)
15       end,
16       ledState, timerSig);
17   Io:digOut(boardLed, ledSig)
18 )
```

[2] http://www.harbaum.org/till/nanovm/index.shtml

```
20 fun setup() =
21   Io:setPinMode(boardLed, Io:output())

23 fun main() = (
24   setup();
25   while true do
26     blink()
27   end
28 )
```

ArduinoBlocks is a drag and drop programming language, similar to the language Scratch, wherein the user writes programs by grouping "blocks" together to form functions. It provides a simple graphical user-interface and functions for various add-on modules for the Arduino, and the user isn't required to write much when programming in ArduinoBlocks.

Figure 2.1 shows how the programming interface for ArduinoBlocks looks.

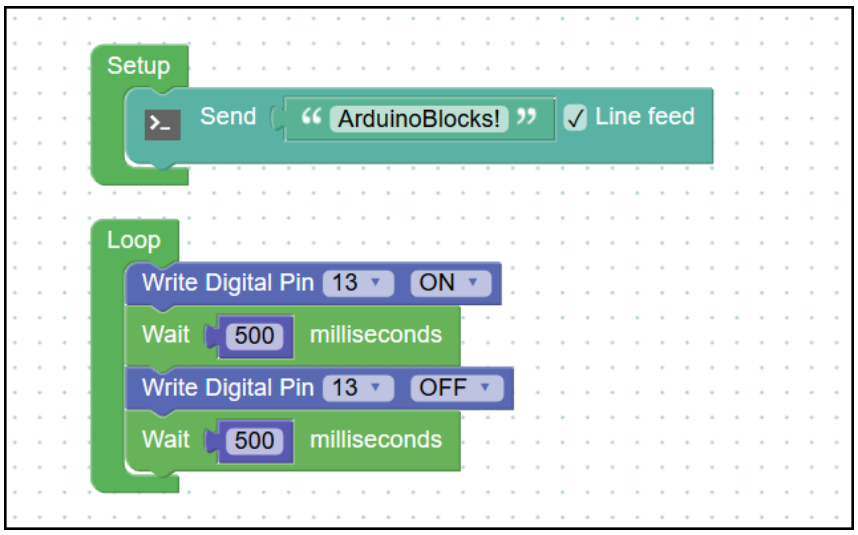


Figure 2.1: The default code when opening ArduinoBlocks

These are, of course, not all the available languages for programming for Arduino. The project group is also aware of a language called DK-BASIC, a subset of BASIC designed to work on Arduino, but that language is still in its alpha stage, so it won't get more than a mention here.

3 Theory

Implementation 4

5 Conclusion

5.1 Discussion

5.2 Conclusion

5.3 Future work

Bibliography

[1] BusinessInsider, "Denmark is growing into a european leader in software development - 19,000 it and electronics specialists needed by 2030," http://nordic.businessinsider.com/denmark-growing-into-a-european-leader-in-software-development----19000-it-and-electronic 2016, [Online; Accessed: 18-02-2018].

[2] Arduino, "Frequently asked questions," https://www.arduino.cc/en/Main/FAQ/#toc2, 2016, [Online; Accessed: 19-02-2018].

[3] Lifelong Kindergarten Group at the MIT Media Lab, "Scratch website," https://scratch.mit.edu/about, 2018, [Online; Accessed: 17-02-2018].

[4] Google Developers, "Introduction to blockly," https://developers.google.com/blockly/, 2018, [Online; Accessed: 17-02-2018].

[5] YoYo Games, "Game maker," https://www.yoyogames.com/gamemaker, 2018, [Online; Accessed: 17-02-2018].

[6] ——, "Gml," https://docs.yoyogames.com/source/dadiospice/002_reference/001_gml%20language%20overview/index.html, 2018, [Online; Accessed: 17-02-2018].

[7] Python Software Foundation, "Python website," https://www.python.org/, 2018, [Online; Accessed: 17-02-2018].

[8] ——, "The python standard library," https://docs.python.org/3/library/index.html, 2018, [Online; Accessed: 17-02-2018].

[9] Randall Munroe, "Python on xkcd," https://xkcd.com/353/, 2018, [Online; Accessed: 17-02-2018].

[10] Wikipedia; Various authors, "Single-board computer," https://en.wikipedia.org/wiki/Single-board_computer, 2018, [Online; Accessed: 18-02-2018].

[11] ——, "Raspberry pi," https://en.wikipedia.org/wiki/Raspberry_Pi, 2018, [Online; Accessed: 18-02-2018].

[12] The LEGO Group, "Lego mindstorms," https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313, 2018, [Online; Accessed: 18-02-2018].

[13] Wikipedia; Various authors, "Educational robotics," https://en.wikipedia.org/wiki/Educational_robotics, 2018, [Online; Accessed: 18-02-2018].

[14] Arduino, "Arduino reference," https://www.arduino.cc/reference/en/, 2018, [Online; Accessed: 25-02-2018].

[15] Juniper-Lang, "Juniper programming language tutorial," https://www.juniper-lang.org/turorial.html, 2018, [Online; Accessed: 25-02-2018].

Part I

Appendix