School of Electronic Engineering and Computer Science

MSc <Computer Science>
Project Report 2019

< A Domain Specific Language for programming FRDM-KL25Z –Special for child programming >

<Peiling Yi>

Queen Mary
University of London

August 2019

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr Matthew Huntbach, who I have sought for advice and guidance in this dissertation development. He is a respectable and resourceful scholar. Without his instruction, kindness and patience, the project would have not been the same. Thank you for every effort he made for me in helping me during the period of project.

A thankful note of appreciate is given to the team of Electronic lab for fully supporting of the borrowed equipment.

I should extend my thanks to all lecturers, who leading me to develop the fundamental academic ability in the two academic years.

Lastly, I need to thank my family and friends for their full support of my study in these two years.

Abstract

Due to the limited existence of textual programming language designed specifically for children. This project will implement a textual domain specific language called ZYX, which is designed for primary school children to program ARM embed board. ZYX keeps basic expression and statements in the general programming language. However, simplify the abstract syntax and complex programming concepts for increasing productivity and speed the software development process. ZYX has independent syntax and compiler, which will interpret the ZYX source file to C++/C source file. The C++/C code is specific for ARM embedded board SDK. ZYX 's syntax is totally different with its' host language Haskell, which is for coding ZYX compiler. For validating the availability and reliability of ZYX, white box test, and black box test are used. The black box test is operated by children who is the end-user of ZYX. The testing ARM embedded board in the project is FRDM-KL25Z.

ZYX will be evaluated by children through a serial of experiments to see if the language can be easy mastered and help children to understand general programming language. The attempt is a good reference for developing other DSL for children.

Keywords: Textual domain specific language, compiler, children, Haskell, C++/C, Black box test, White box test, Evaluating.

Table of Contents

Acknowledgements 1

Abstract 2

1.Introduction 5

  1.1Overview 5

  1.2 Motivation 6

  1.3 Objectives 6

  1.4 Contributions 8

  1.5 Outline 9

2.Background 9

  2.1 Programming technologies and tools for children 9

    2.1.1 Scratch 10

    2.1.2 Blockly 11

    2.1.3 BBC micro-bit 12

  2.2 Learning curve and National curriculum in England 13

  2.3 The Domain-specific languages engineers 13

  2.4 Parser combinator 14

  2.5 ARM embedded platform 15

3. ZYX specification 15

  3.1 Runtime environment 15

    3.1.1 The editor for coding 15

    3.1.2 Operation platform 16

  3.2 Language specification 17

    3.2.1 Structure of line 17

    3.2.2 Structure of block 18

    3.2.3 Reserve words 18

    3.2.4 Structure of expression 19

    3.2.5 Structure of Statement 19

    3.2.6 Structure of the method 20

    3.2.7 Indentation 20

    3.2.8 File encoding 20

    3.2. 9 Blank Lines 20

    3.2.10 Whitespace 20

    3.2.11 Variable Names 20

    3.2.12 The sample of program 21

4. The features of ZYX 21

4.1 The features of Domain 21
4.1.1 User Profile..... 21
4.1.2 ARM FRDM-KL25Z 22
4.1.3 Domain Model 22
4.2 The features of language 23
4.2.1 The features of ZYX Syntax..... 24
4.2.2 The features of compiler..... 26
4.2.3 The features of code generation..... 29
5. The implementation of XYZ..... 30
5.1 Code structure 30
5.2 Data structure tree implementation..... 31
5.3 Tokenization..... 35
5.4 Data type checking..... 42
5.5 Interpretation..... 43
5.6 Code generation..... 44
6.Test..... 46
6.1 White box test..... 46
6.1.1 Statement and Branch Coverage test 46
6.1.2 Unit coverage test..... 47
6.1.3 Black box test 49
7.Evaluation 53
7.1 Experiments general procedure 53
7.2 Materials for the experiments 55
7.3 Equipment and environment of experiment 55
7.4 Feedback 56
8. Discussion and Conclusions 57
9.Further work 58
10.Reference..... 59
10.1 Embedded platform..... 59
10.2 Children programming languages and tools..... 59
10.3 Research papers and books 59
10.4 Programming language 60
10.5 Tools..... 60
A. Appendix..... 61
A.1 Questionnaire 61
A.2 Slides..... 65

A.3 Part of Measurement sheet..70

1.Introduction

1.1Overview

Today, we are witnessing an increasing use of embedded systems in the aerospace industry, telecommunications, automotive, chemical, civil, energy, medical, manufacturing, transportation, entertainment, and education.[1] Typically, users from different backgrounds are involved in the development of these embedded systems, ranging from sophisticated and in-depth computer scientists to domain experts and even end users such as children. The complexity of the system is increasing and is heavily dependent on its actual operational domain. As a result, some interface languages are designed to suit different user profiles. For example, Arm company use C/C++ as user interface language to help developer to code the embedded board efficiently.[2]

Due to many well-succeeded achievements, the embedded system is now considered as a good platform and medium to introduce children about computer science knowledge and programming concepts. Some of its advantages are obvious. For example, the embedded board's exterior helps children easy to understand concept of the computer hardware. Another good example is children can easily observe how their code reacts on hardware. Their communication has become more frequent. it will be a great encouragement for their programming enthusiasm, such as simply flashing an LED light on a board.[3]

If children have an opportunity to use “simplify” but "proper" textual programming language to develop a series of program in embedded system, and observe the code they developed, which will be a great learning process.

1.2 Motivation

The country will always need software and hardware engineers —great engineers!

At present, Children's programming development platform is still based on graphical programming language. Part of the reason for this is that young children still have difficulty understanding the complex syntax in the current textual programming language. However, there is a big gap between graphical programming languages and general-purpose programming languages. [4]

If a suitable textual programming language for children is designed, which hides some abstract and complex programming concepts and syntax according to children’s understanding and cognitive ability but keep the general structure of textual programming language, it will be a positive entry point to help children learning serious computer science knowledge and speeding the programming skill development. At the same time, the experiments of organising children to use the textual language for children will be a good resource to develop other textual DSL for children in future.

1.3 Objectives

In this project, A new textual programming language for children is created and put it in some experiments to test if it is practicability and attractive for children. For reaching the aim, three sub-goals are set as following.

- **Building The domain model**

ZYX is a textual domain specific language, which is for programming ARM Board. ARM offers uniform SDK for all boards based on 32-bit ARM Cortex-M microcontrollers. In the project, the main test board is FRDM-KL25Z. In theory, if the program can run on FRDM-KL25Z, then it can run on all other the same processor ARM board.

The target user for ZYX is primary school children with ages starting from 9 years old. ZYX is intended to be used in education fields.

ZYX can program different behaviours of FRDM-KL25Z.

- **The implementation and testing of ZYX**

Implementing ZYX requires three main tasks. ZYX syntax definition**,** ZYX compiler and C ++ /C file generation.

Testing case library is built for testing the flexibility and extensibility of ZYX. White box test and Black box test are used in testing ZYX. White box test includes statement coverage test and unit coverage test. Black box test is operated by end-user.

- **ZYX Evaluation**

Evaluation is the most important task to do in the project. It is necessary to prove that the ZYX is improving the end-user programming process in the domain area and reach the goals. Therefore, observing the behaver of embedded board based on ZYX code written by children is the main stage of evaluation. A serial recorded experiments' data was analysed to see if ZYX is easy to learn and improve the speed at which children learn general programming language.

1.4 Contributions

Currently, there are many visual and block-based programming languages designed for children. For example, Scratch and Blockly. These languages tried to simplify the programming process. Thus, children they can learn other languages starting with a bit more experience. The main feature of these kind language is avoiding syntax and punctuation of traditional textual programming language, which can speed the target user to build their application. However, at the same time, the design deprives children of opportunity to make rapid progress in the future according to “learning curve theory” [8]. These languages use a lot of natural language to simplify the syntax of traditional programming language, which can cause a lot confusion when children learn general programming language.

In 2016, BBC sent a pocket-sized codable computer micro-bit to teachers for the first time to encourage children to actively learn programming. It presents that the combination of hardware and software is the development trend in the field of education. Nevertheless, this product still does not solve the issue about learning curve. the bridge from graphic language to general programming language has not yet been built.

This project will be an interesting attempt to make up for the above deficiencies. A new textual Domain Specific Language ZYX has been built to encourage children to code ARM embedded board. ZYX has kept the main general textual language common syntax but hiding some C++/C ambiguous syntax. Therefore, the children will gradually build up the fundamental knowledge of general programming language, sequentially, quickly master any other programming language they will use in future. At the same time, the feedback of experiments will be a good reference for further developing other textual programming languages for children in different domain.

1.5 Outline

The dissertation is organized in ten sections

1. Chapter 2 will discuss the whole background of the project, which include The Domain-specific languages engineers, programming language used in the project, embedded platform and technique of compiling and so on. Particularly, compare these specific programming languages and platform for children.

2. Chapter 3 gives the language ZYX specification.

3. Chapter 4 provides the features of ZYX.

4. Chapter 5 describes implementation of ZYX.

5. Chapter 6 describes the test process and part of test case.

6. Chapter 7 presents the details of evaluation for ZYX.

7. Chapter 8 is for discussion and conclusion about the project.

8. Chapter 9 is for introducing further work.

9. Chapter 10 shows all references and divides them into five categories.

10. Chapter A is Appendix, which includes education slide and questionnaires.

2.Background

2.1 Programming technologies and tools for children

Children is a natural explorer and fast learner. Especially the attitude to new technology is impressive. Nowadays, many programming languages and products are developed for speeding children's learning rate. In this chart, the features of some trending products in the fields are explored.

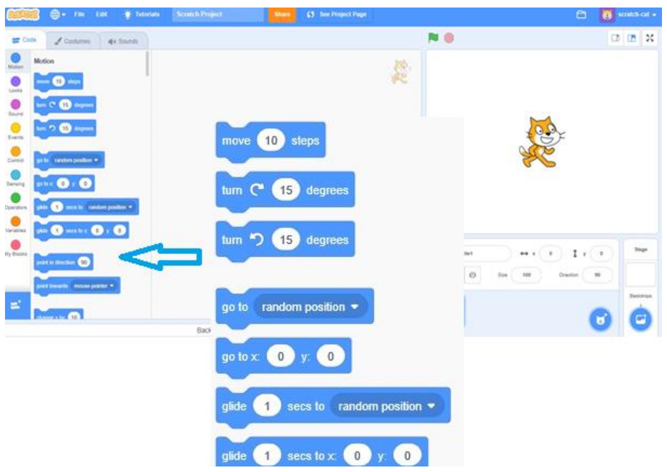
2.1.1 Scratch

Scratch is perhaps the most known visual programming language for children programming education. Scratch syntax is based on a graphic set of "programming modules" that aim to teach basic programming concepts. It allowed user to create programs by manipulating program elements graphically, rather than specifying them by text.

Figure2.1 is the Scratch3.0 interface. The Scratch interface has three main parts: a staging window, a panel, and a coding window for placing and arranging blocks into runnable scripts. [5] It allows users to drag and drop program elements into the coding window, which is the main approach to do coding work. The vast number of natural languages are used to explain the functionality of each block makes it far from the syntax of common programming languages such as C++, Java, or python. The Scratch seems like a successful tool to encourage kids to be more creative, but it's far from an introductory programming language.

Figure 2.1 the interface of Scratch 3.0

(https://scratch.mit.edu/projects/editor/?tutorial=getStarted**)**

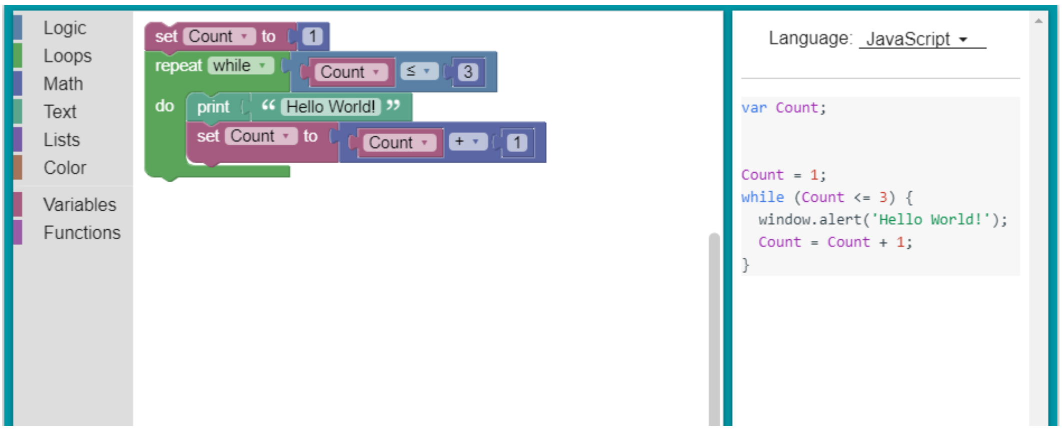


2.1.2 Blockly

Blockly is another block-based visual programming language. It is a project of Google and runs in a web browser. It has a similar look to Scratch but a completely different design goal. [6] Actually, it is a graphical programming language that is made up of general programming language. It is more commonly described as “java script library”. Blockly combine the multi-shaped and multi-coloured block with natural language to help children to learn the concepts of programming language. It can generate several programming languages codes. This kind of design idea makes it will have an excellent prospect. However, too much comprehensive words using in the block give children the wrong idea of programming and being too "sugar-coated" or watered-down.

Figure 2.2 The interface of Blockly

(https://developers.google.com/blockly/**)**



```
set Count to 1
repeat while Count ≤ 3
do print "Hello World!"
   set Count to Count + 1
```

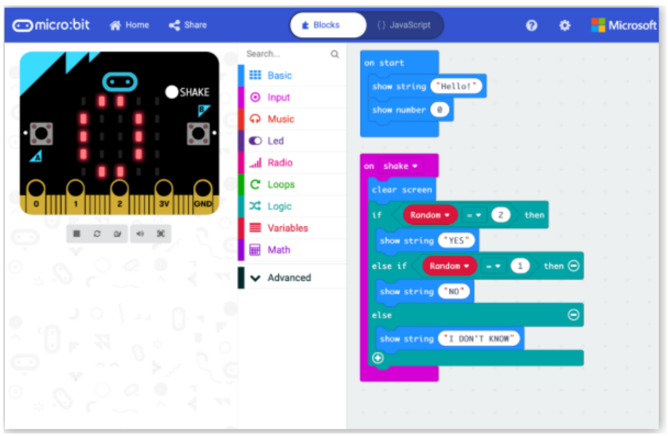
```
var Count;


Count = 1;
while (Count <= 3) {
  window.alert('Hello World!');
  Count = Count + 1;
}
```

2.1.3 BBC micro-bit

BBC Micro Bit is designed by BBC for computer science education in the UK. It provides a platform to program ARM-based embedded system.[7] This product reflects the development trend of compute education ,and encourage student to self-directed learning the computer science fundamentals skill by using the characteristic of easy interaction of embedded system. This is an attractive education product that integrated with a lot of interesting components. However, the idea of teaching programming has not changed too much.

Figure 2.3 The interface of BBC micro-bit

(https://microbit.org/code/**)**



```
on start
  show string "Hello!"
  show number 0

on shake
  clear screen
  if Random = 2 then
    show string "YES"
  else if Random = 1 then
    show string "NO"
  else
    show string "I DON'T KNOW"
```

2.2 Learning curve and National curriculum in England

The learning curve was first described by Herman Ebbinghaus in 1885 in the field of learning psychology. The concept suggests that for activities that are difficult to learn, although the learning curve is steep at first, there will be rapid progress after that.[8]

the Industry offers too much sugar-coated programming learning experience at the outset, which is good for selling its products, but it is still open to debate whether it’s good for children to learn the technical skill.

The UK government has published a national curriculum in England on computing programmes of study. [9] According to the purpose of study, they detailed the learning objectives of different key stage level learning target. This will be the key guide for ZYX design.

2.3 The Domain-specific languages engineers

A domain-specific language (DSL) is kind of computer languages, which specialized to some applications in domain rather than a general-purpose language that is aiming to any kind of software problems. DSLs are very common in the fields, which ranging from widely used language for the common domain, such as SQL for database, CSS for webpage and regular expression for string searching algorithm.[12]

The value of DSLs is that well-designed DSLs are easier to program than traditional libraries. This greatly improves programmer productivity, especially communication with domain experts.

Generally, there are two main ways to implement DSLs. One is called External DSLs, which has an independent interpreter and compiler. ZYX is an example of external DSL, which has standalone complier to translate the ZYX code to C++/C code and its syntax totally independent of its host language Haskell. Using external DSL means it is processed as standalone tools, called via direct user operation by command line. The key advantage of an external DSL is the freedom to choose any form you like. Therefore, the domain can be expressed in its simplest form for reading and modification. The format is limited only by the ability to build interpreter. The interpreter can parser the configuration file and generates an executable file.

Another category is known as Internal DSLs, which typically implemented within a host language as a form of API and tended to be limited to the syntax of the host language. One obvious advantage for this kind of DSL is that there is no need to build a compiler. but the resulting limitations on the DSL itself can also greatly reduce its advantages. [10]

2.4 Parser combinator

Currently, there are generally two different ways to write a parser: parser generator and parser combinator. Which method to use depends on the ease of the syntax and the speed of parsing. In the case, the grammar is very simple and context-free, so we can choose either LL parsing or LR parsing. According to the situation, parser combinator maybe is a better option. [11] The parser combinator is a higher-order function that takes multiple parsers as parameters and returns a new parser as output. Another very important reason, Monads, Functors and Maybe data type in Haskell are so attractive to build parser.

2.5 ARM embedded platform

Today, the ARM family accounts for 75 percent of all 32-bit embedded processors, making it one of the largest 32-bit architectures in the world. ARM processors are found in many consumer electronics products, from portable devices (mobile phones, multimedia players, handheld video games, and computers) to computer peripherals (hard drives, desktops, routers) and even in military installations such as missile-borne computers. [12]

Mbed is a set of tools provided by ARM company for rapid Development of application prototype of ARM architecture, including Software Development Kit (SDK), Hardware Development Kit (HDK) and web-based online compilation environment (Mbed online Compiler).[15]

Mbed SDK is a software development framework to C/ C ++. In this project, the final output code is C++/C that calls the Mbed SDK.

3. ZYX specification

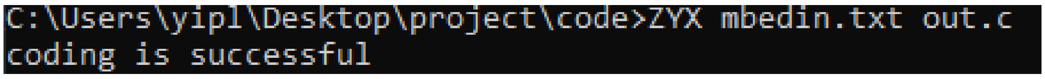
3.1 Runtime environment

3.1.1 The editor for coding

ZYX code can be edited by any text editor.

ZYX compiler is processed as standalone tools, called via direct user operation by command line. The compiler ZYX.EXE can take two parameters. One is the input file name, another one is the C++/C source file, which is generated by compiler.

Figure 3.1 The command line for executing ZYX compiler



```
C:\Users\yipl\Desktop\project\code>ZYX mbedin.txt out.c
coding is successful
```

If the compiler cannot find any parameters. The default file names are used, which are mbedin.txt and mbedout.c.

3.1.2 Operation platform

Creating a program for Arm board is not one step job. Arm company now only provides online compiler for compiling the C++/C code to machine code, so after using ZYX compiler to convert ZYX code to C++/C code, another two steps need to be done. One is copying the code to Mbed platform for complying the C++/C code to machine code , another is for copying the machine code to the directory of FRDM-KL25Z in the compute which connect to FRDM-KL25Z board. The figure 3.2 shows the whole procedure of programming ARM embedded platform by ZYX.

Figure 3.2: The process of using ZYX

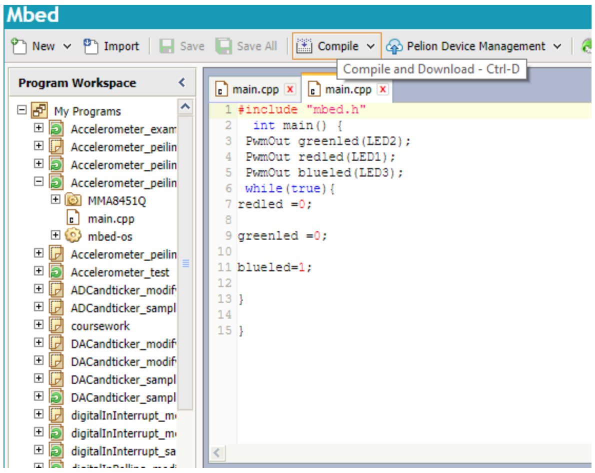
Order	User usage process	ZYX compiler processing process
1	Open a text editor	
2	Users write ZYX source file.	
3	Users execute ZYX compiler to convert ZYX source file to C++/C source file. The compiler gives execution information.	Generating the C/C++ code file Providing a simple syntax hint if the text file cannot be parsed
4	Users Coping the code to Mbed platform for complying to machine code	

5	Connect ARM embedded board with the USB port of computer by cable.	
6	If compiling is successful, copy the machine code to the memory of board of FRDM-KL25Z.	

Figure 3.3 is Mbed online platform. The C++/C code generated by ZYX compiler is copied to the platform and click “compile”, then the machine code is automatically download to the user computer. Since the board support USB controller, so just copy the final machine code to the directory of embedded board in user computer.

Figure 3.3 The Mbed platform

(https://os.mbed.com/handbook/mbed-Compiler)



```
#include "mbed.h"
 int main() {
PwmOut greenled(LED2);
PwmOut redled(LED1);
PwmOut blueled(LED3);
while(true){
redled =0;

greenled =0;

blueled=1;

}

}
```

3.2 Language specification

3.2.1 Structure of line

A ZYX program is divided into some physic lines and every physic line is a sequence of characters terminated by end-of-line symbols (\r\n or \n in windows).

3.2.2 Structure of block

Using the curly braces of Allman style to indicate blocks of statement codes, even if it’s only one line.

3.2.3 Reserve words

These reserve words will be parsed to the reserve words of C++/C. the following table is a comparison between these two languages.

Figure 3.4 ZYX reserved words versus C++/C reserved words

Word Of DSL	**Word of C++/C**
Wait=	wait ()
True	true
False	false
Redled	LED1
Greenled	LED2
Blueled	LED3
Touchsensor	TSISensor
Acc	MMA8451Q(PTE25, PTE24, MMA8451_I2C_ADDRESS)
On	0
Off	1
Red	1
Green	2
Blue	3
And	&&
Or	||
>	>
<	<
+	+
-	-
Eq	==
If	if
While	while
Switch	switch
Case	case

Break	break

3.2.4 Structure of expression

Figure 3.5 Structure of expression

The type of expression	Format
Hardware object assignment	Variable name= Object name Object name is one of [Redled, Blueled,Greenled,Acc,Touchersens or]
Variable assignment,	Variable name=value
Boolean Expression	True/False
String expression	Specifically, for the statement of Switch …Case …
Wait expression This is for thread.wait	Wait= digital number (int /float)

3.2.5 Structure of Statement

Figure 3.6 Structure of statement

The name of statement	Format

If	If([“condition1]and/or [condition2] and / or [condition n])
	{[expressions]}

While	While(condition)
	{programs}

Switch Case	Switch(state)
	{
	Case state value 1:
	Expressions
	Break
	…
	Case state value n:
	Expressions
	Break
	
	}

3.2.6 Structure of the method

Custom functions are not supported by compiler, but it can call pre-defined internal function. The internal function only combined with object.

Figure 3.7 Structure of method

Method of DSL	Method of C++/C
X	getAccX ()
Y	getAccY ()
Z	getAccZ ()
D	readDistance ()

3.2.7 Indentation

Use spaces as indentation, don’t use tab as indentation. Otherwise, sometime will give compiler error.

3.2.8 File encoding

Code in the source file should always use UTF-8.

3.2. 9 Blank Lines

Only one blank line be allowed using for separating each code line. No blank line between each code line is recommended.

3.2.10 Whitespace

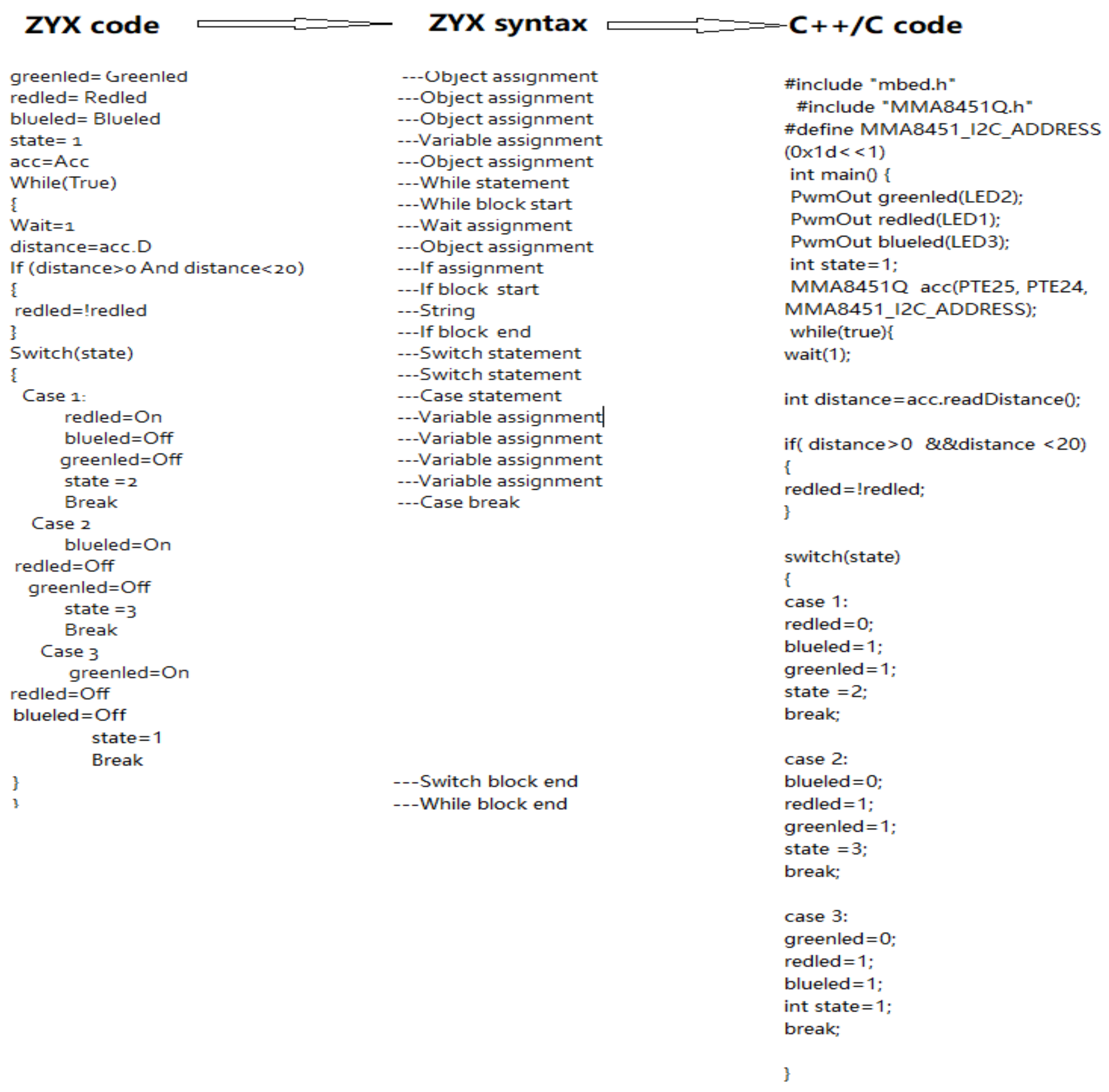
Using whitespace will not affect any results of compiling.

3.2.11 Variable Names

Variable name should be lowercase, numbers or underscores are unacceptable. ZYX compiler will add the C++/C datatype to variable dynamically.

3.2.12 The sample of program

Figure 3.8 The sample of program


ZYX code
ZYX syntax
C++/C code

```
greenled= Greenled                           ---Object assignment
redled= Redled                               ---Object assignment
blueled= Blueled                             ---Object assignment
state= 1                                     ---Variable assignment
acc=Acc                                      ---Object assignment
While(True)                                  ---While statement
{                                            ---While block start
Wait=1                                       ---Wait assignment
distance=acc.D                               ---Object assignment
If (distance>0 And distance<20)              ---If assignment
{                                            ---If block  start
 redled=!redled                              ---String
}                                            ---If block  end
Switch(state)                                ---Switch statement
{                                            ---Switch statement
  Case 1:                                    ---Case statement
         redled=On                           ---Variable assignment
         blueled=Off                         ---Variable assignment
        greenled=Off                         ---Variable assignment
         state =2                            ---Variable assignment
         Break                               ---Case break
   Case 2
         blueled=On
 redled=Off
   greenled=Off
         state =3
         Break
    Case 3
          greenled=On
redled=Off
blueled=Off
              state=1
              Break
}                                            ---Switch block end
}                                            ---While block end
```

```
#include "mbed.h"
  #include "MMA8451Q.h"
#define MMA8451_I2C_ADDRESS
(0x1d<<1)
 int main() {
 PwmOut greenled(LED2);
 PwmOut redled(LED1);
 PwmOut blueled(LED3);
 int state=1;
 MMA8451Q  acc(PTE25, PTE24,
MMA8451_I2C_ADDRESS);
 while(true){
wait(1);

int distance=acc.readDistance();

if( distance>0  &&distance <20)
{
redled=!redled;
}

switch(state)
{
case 1:
redled=0;
blueled=1;
greenled=1;
state =2;
break;

case 2:
blueled=0;
redled=1;
greenled=1;
state =3;
break;

case 3:
greenled=0;
redled=1;
blueled=1;
int state=1;
break;

}
```

4. The features of ZYX

4.1 The features of Domain

4.1.1 User Profile

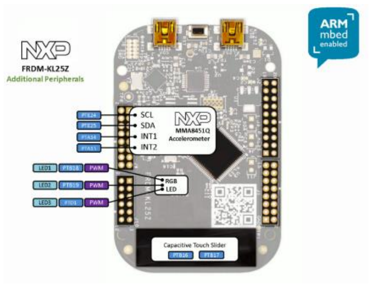
Figure 4.1 the User profile

Characteristics	Content
Age	9 - 11 years old
Academic Year	Year 4 – Year 6
First Language	English
ICT Level	Above the average level from the school
Computer Knowledge	Intermediate
Previous Programming Skill	No
Mathematics Grade	Above the average level from the school

4.1.2 ARM FRDM-KL25Z

ARM frdm-kl25z was selected as the test board of this project. The ZYX will support programming the Touch Sensor, LEDs and Accelerometer.

Figure 4.2 ARM frdm-kl25z

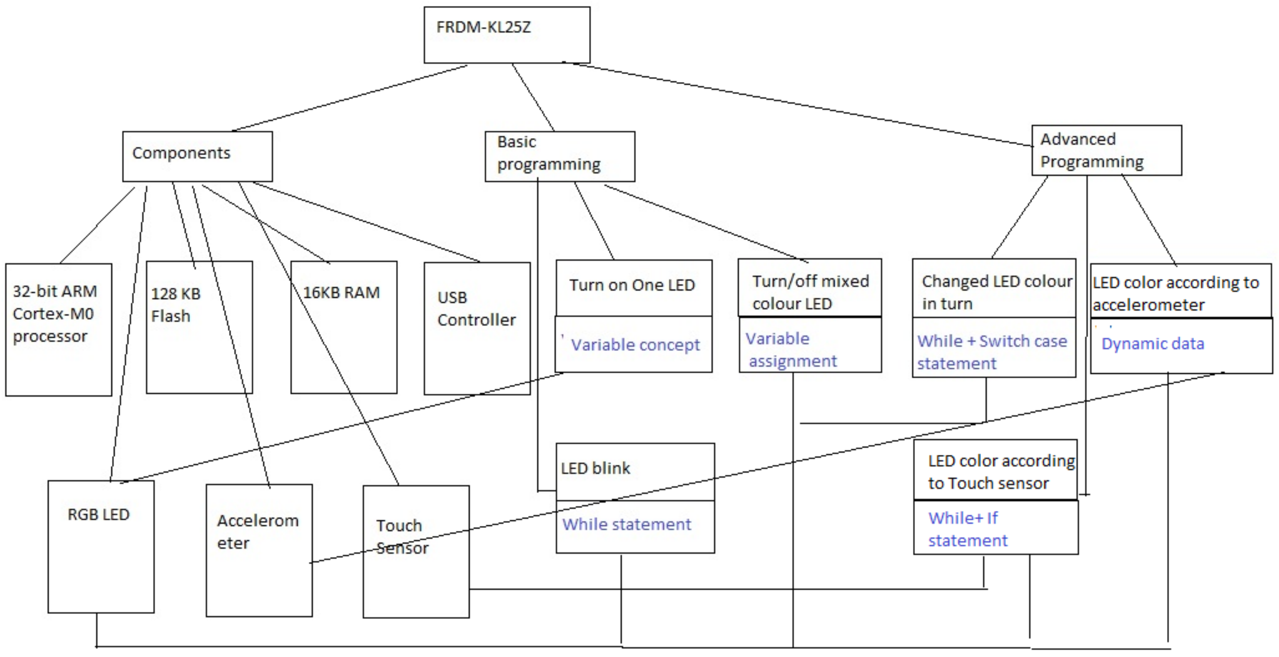


4.1.3 Domain Model

The domain model combined the details analysis of ARM board technologies, Mbed platform and current programming language for children. Once the domain model was completed, the basic feature of ZYX is determined.

Figrue 4.3 presents the domain model. Eight board components are used in programming by ZYX. ZYX is tested to support to program at least six behaviours of ARM FRDM-KL25Z board. Each behaviour will introduce one of general programming concept.

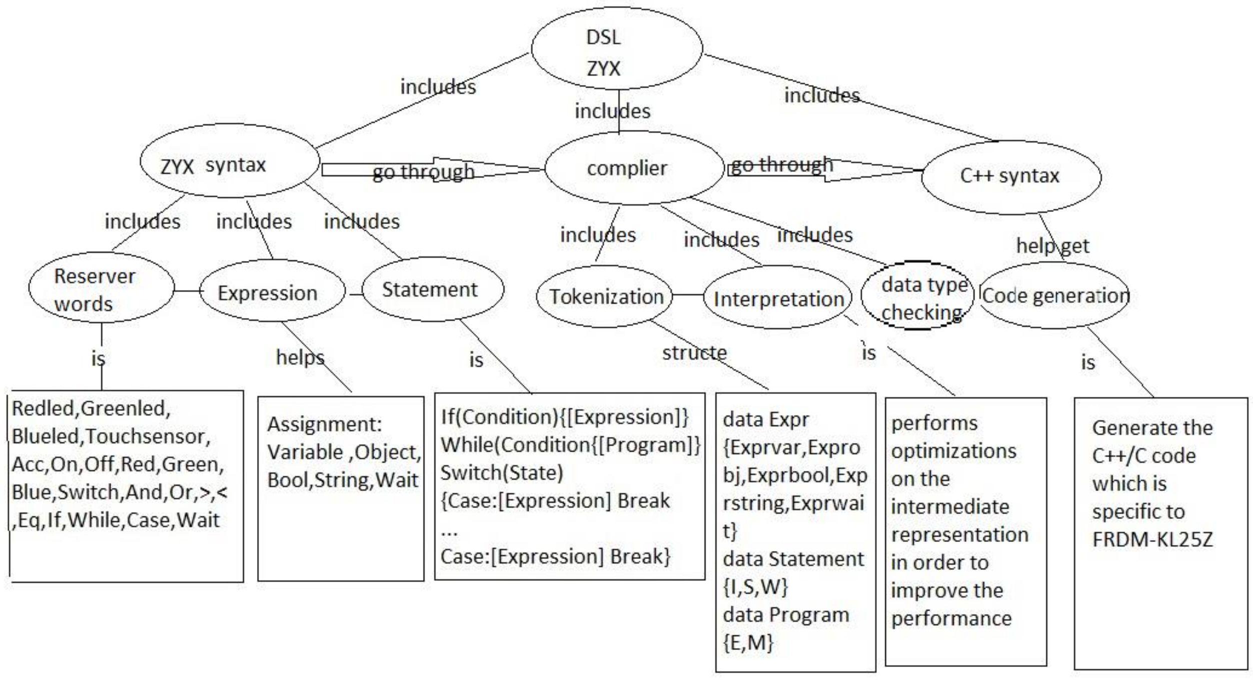
Figure 4.3 The domain model



4.2 The features of language

For giving a clear outlook of the language, all features mentioned below are combined to be a concept map. The details about how to implement these features will be introduced in chapter 6.

Figure 4.4 concept map of ZYX.


DSL ZYX
includes
includes
includes
ZYX syntax
go through
complier
go through
C++ syntax
includes
includes
includes
includes
includes
includes
help get
Reserver words
Expression
Statement
Tokenization
Interpretation
data type checking
Code generation
is
helps
is
structe
is
is
Redled,Greenled, Blueled,Touchsensor, Acc,On,Off,Red,Green, Blue,Switch,And,Or,>,< ,Eq,If,While,Case,Wait
Assignment: Variable ,Object, Bool,String,Wait
If(Condition){[Expression]} While(Condition{[Program]} Switch(State) {Case:[Expression] Break ... Case:[Expression] Break}
data Expr {Exprvar,Exprobj,Exprbool,Exprstring,Exprwait} data Statement {I,S,W} data Program {E,M}
performs optimizations on the intermediate representation in order to improve the performance
Generate the C++/C code which is specific to FRDM-KL25Z

4.2.1 The features of ZYX Syntax

The definition of ZYX syntax is for increasing productivity and simplify the software development process. The syntax ought to include the following features:

Feature1: Loose grammar

For example: no data types, no constant, no ";" at the end of line.

Feature 2: Hide the hardware component definition and SDK

For example: no abstract object definition, no library included, no main function adding.

Feature 3: Keep the fundamental programming concept.

For example: Logical operator, Variable assignment, If statement, While statement, Switch Case statement and Object assignment.

Feature 4: Replace abstract operator or symbol in C++/C to be readable operator or symbol in ZYX code

For example: “&&” in C++ /C is replaced by “And” in ZYX

Feature 5: Flexibility and expansibility

Users can create from one-line programs to complicated program by Using ZYX. Figure 4.5 shows a basic program to learn about the variable and object assignment by switching on a green LED of board and a complicated program to learn statement by control LEDs according to the value of touchpad.

Figure 4.5: DSL versus C++/C

DSL code	C++/C code

greenled=Greenled
greenled=On
	#include “mbed.h”
	int main() {
	PwmOut greenled(LED2);
	greenled=0;
	}

led1=Greenled
led2=Redled
acc=Acc
touch=Touchpad
While(True)
{
led1=acc.X
led2=acc.Y
	#include “mbed.h”
	#include “MMA8451Q.h”
	#define MMA8451_I2C_ADDRESS (0x1d<<1)
	#include “TSISensor.h”
	int main() {
	PwmOut led1(LED2);
	PwmOut led2(LED1);

distance=touch.D
 	
If (distance > 0 And distance<20)
 	
{
 	
led1=!led1
 	
}
 	
If (distance >20 And distance<40)
 	
{
 	
led2=!led2
 	
}
 	
}
	MMA8451Q acc(PTE25, PTE24,
	
	MMA8451_I2C_ADDRESS);
	
	TSISensor touch;
	
	while(true){
	
	led1=1.0-abs(acc.getAccX());
	
	led2=1.0-abs(acc.getAccY());
	
	int distance=touch.readDistance();
	
	if (distance >0 && distance <20)
	
	{
	
	led1= !led1;
	
	}
	
	if(distance >20 &&distance <40)
	
	{
	
	led2=!led2;
	
	}
	
	}
	
	}

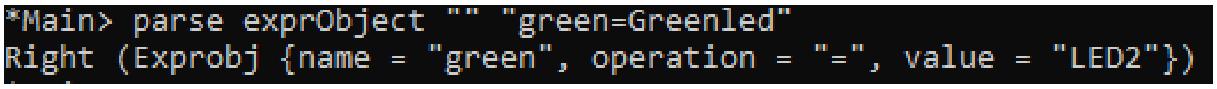
4.2.2 The features of compiler

Basing on previous syntax definition, the key stage of implementing a ZYX is compilation method, which is able to parse a program written by ZYX and process it for generating the code in C++/C language. This process can be divided into several stages.

- **Tokenization**

The string stream is decomposed into tokens according to the definition of language syntax. Each token is a single atomic element of the ZYX. The token can be a keyword, variable name, variable value, object name or symbol.

Figure 4.4 The sample of parser exprObject



```
*Main> parse exprObject "" "green=Greenled"
Right (Exprobj {name = "green", operation = "=", value = "LED2"})
```

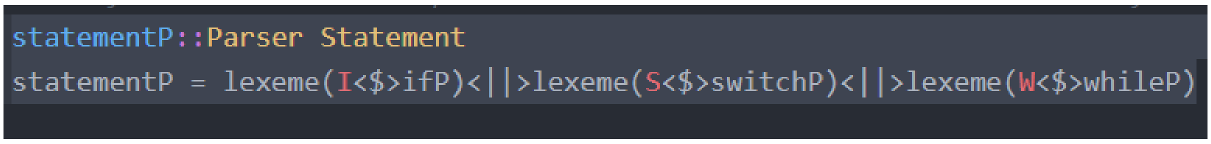
The parse separates the input string *"green=Greenled"* be three tokens *{name = "green", operation = "=", value = "LED2"* according to syntax structure.

There are two main features of ZYX tokenization.

Feature 1: Haskell parsing library Text.ParserCombinators.Parsec is used.

Feature 2: Composability is an obvious feature of the compiler. A big, top and complex parser is construct by combining a serials of small and reusable high-order parser combinator. For example, the parser statement is the result of three parser combinator combined:

Figure 4.5 The code of parser statement



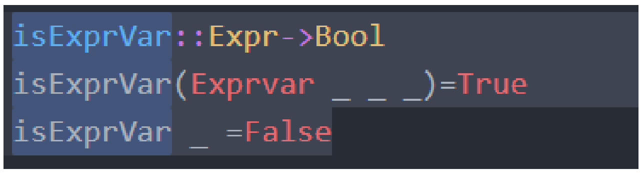
```
statementP::Parser Statement
statementP = lexeme(I<$>ifP)<||>lexeme(S<$>switchP)<||>lexeme(W<$>whileP)
```

- **Data type checking**

Some jobs of validation have done by parser. However, it is not enough, a very important validation job is data type checking. Every data is filtered by data type checking.

For example, the function “isExprVar” is for checking if the data type is ExprVar. If it is not, the parser will skip this part of data to do next.

Figure 4.6 The code of data type checking



```
isExprVar::Expr->Bool
isExprVar(Exprvar _ _ _)=True
isExprVar _ =False
```

- **Interpreting**

Feature1: the interpretation of these tokens according to the syntax of C++/C. All are dynamic transformation.

For example:

Input string	Tokens	C++/C code
"int n=5\n"	[[E (Exprvar {name = "int n", operation = "=", value = "5"})]]	int n=5
” If(a>b And b>c)\n{\n myled=On\n}\n"”	[[M (I (If [IfE (Exprvar {name = "a", operation = ">", value = "b "}),IfE (Exprvar {name = "", operation = "&&",	if(a>b && b>c)\n{\n myled=0\n\n}n”

	value = "b"}),IfE (Exprvar {name = "", operation = ">", value = "c"})] [Exprvar {name = "myled", operation = "=", value = "0"}]))]]	

Feature2: the interpretation of these tokens according to the SDK of mbed. All are dynamic transformation.

For example:

Input string	Tokens	C++ code for SDK
"green=Greenled\n"	{name = "green", operation = "=", value = "LED2"}	PwmOut green (LED2);
"acc=Acc\nled1= acc.X\n"	[[E (Exprobj {name = "acc", operation = "=", value = "Acc"}),E (Exprvar {name = "led1", operation = "=", value = "acc.X"})]]	MMA8451Q acc(PTE25, PTE24, MMA8451_I2C_ADDRESS); led1=1.0-abs(acc.getAccX());

4.2.3 The features of code generation

Feature1: Dynamic include ARM board components library to implement the final C++/C file. For example, if “Acc” is defined in code line written by DSL, the head file "MMA8451Q.h” must be include in the final C++/C file and “MMA8451_I2C_ADDRESS” need to be define.

Feature 2: Dynamic include main function of C++/C

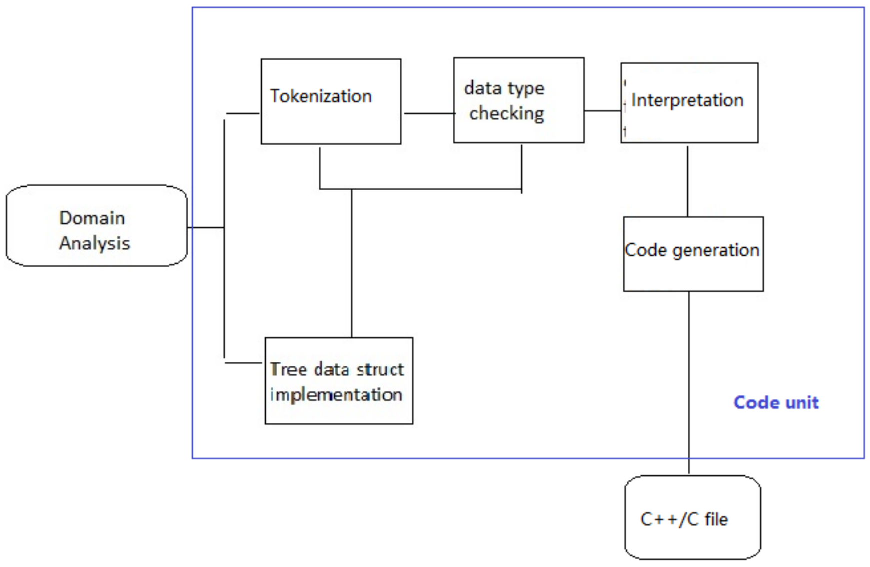
Feature 3: The output filename of C++/C can be defined by end user or using the default filename.

5. The implementation of XYZ

5.1 Code structure

For implementing the DSL code, the parser combinator was created with Haskell parsing library, which generate the collection of tokens. For validating these tokens, the data type checking happens following it. All the tokens are kept in abstract parser tree, so how to extract these data from tree and format it to the C++/C code must be considered carefully. Before generating code, two things need to be designed, one is for dynamic adding head file for Mbed platform and another is for reading input file and writing output file. The file name and type can be defined by end-user. The figure 5.1 shows the structure of the code, every unit includes a set of functions. All units will be detailed in the chapter.

Figure 5.1 The structure of code



5.2 Data structure tree implementation

As previous mentioned, the compiler should execute tokenization to get a collection of tokens. Therefore, a data structure tree is designed for structuring these tokens according to syntax structure tree. Both the parser combinator and data type check fully referenced the data structure tree.

Figure 5.2 shows the inheritance relationship from top to bottom and the explanation of each data type.

Figure 5.2 data structure tree

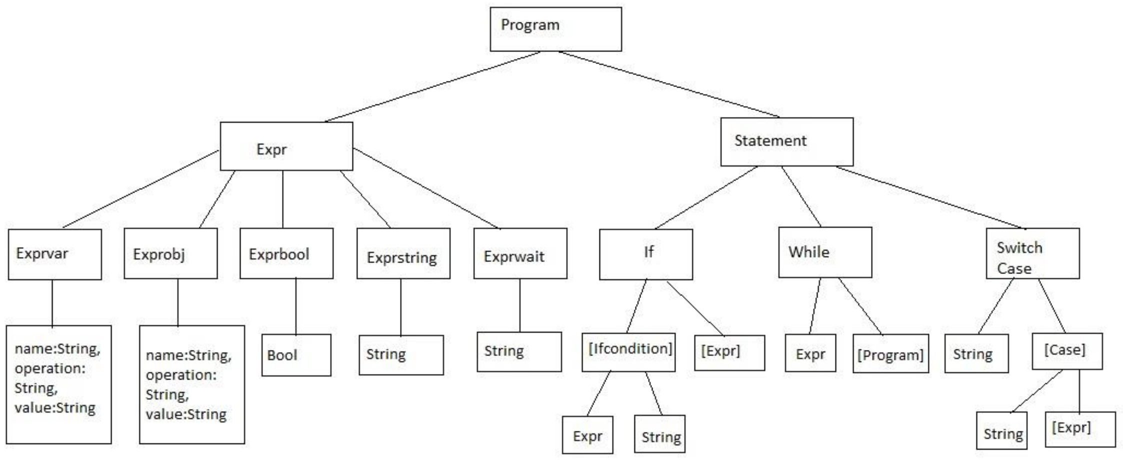


Figure 5.3 The explanation of Data structure

Data type name	definition	Explanation
Expr	Exprvar | Exprobj | Exprbool | Exprstring | Exprwait	datatype Expr is defined for five kinds of expression, which includes five value constructors: Exprobj, Exprvar ,Exprbool, Exprstring and Exprwait
Exprobj	{ name :: Strin , operation::String, value :: String }	For hardware component assignment
Exprvar	{ name :: String ,operation::String , value :: String }	For Variable assignment
Exprbool	Bool	For Boolean expression
Exprstring	String	For switch and case expression
Exprwait	String	For "Wait" expression
Ifcondition	IfE Expr | IfO String	datatype Ifcondition is defined for the condition of If statement, which include two value constructors, IfE or IfO, IfE is

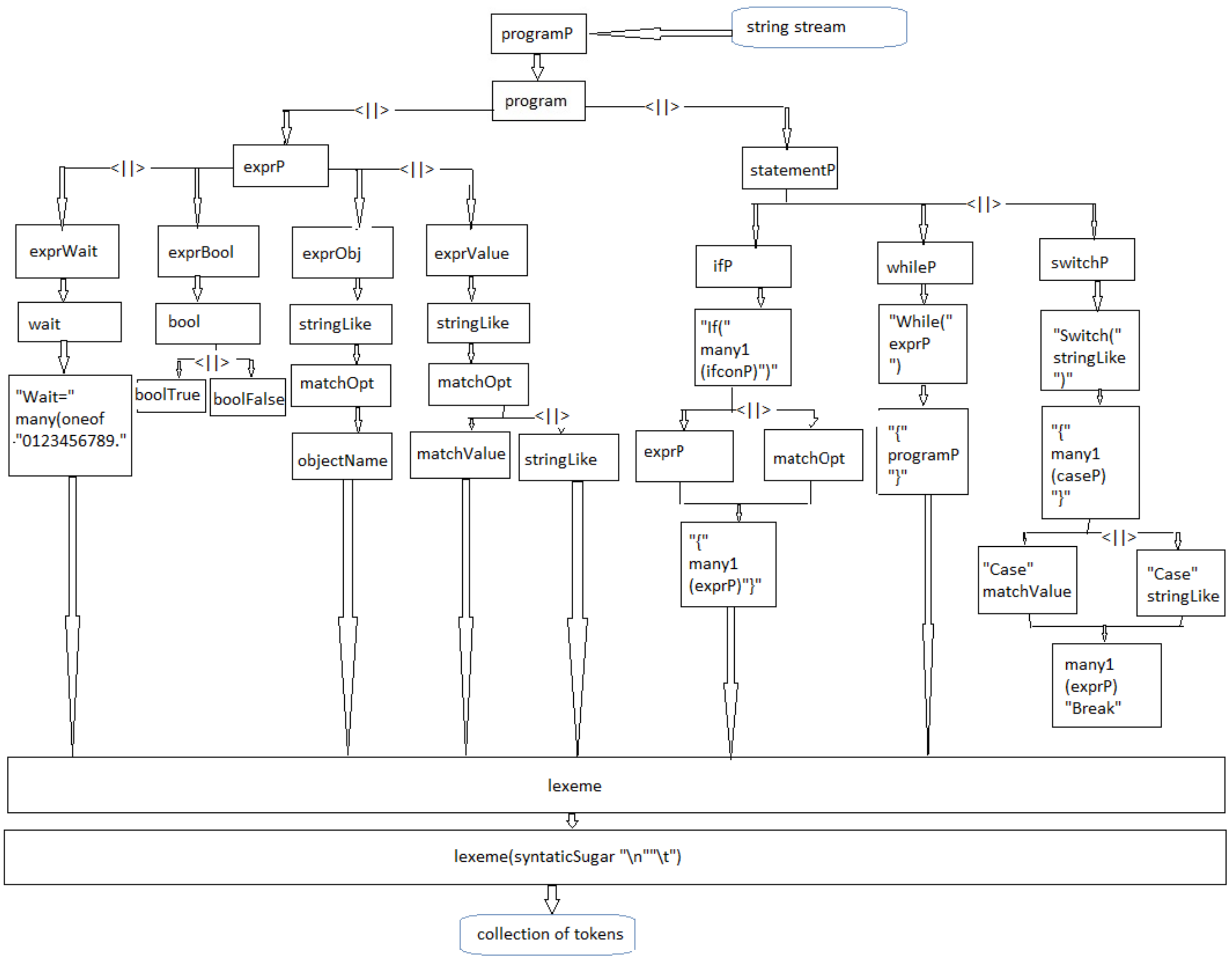
		datatype Expr, IfO is datatype String. As a result, If statement can support (x>y) or (True) Syntax
If	If [Ifcondition] [Expr]	Datatype If is defined for If statement, which includes two fields, one is the array of Ifcondition, another is the array of Expr. As a result, If statement supports two types of syntax, If(x>y){expression} and If(True){expression}
While	While Expr [Program]	While datatype have two fileds, one is datatype Expr and another is the array of program. As a result , While statement supports nest syntax. For example, While(..){While(..){...}}. or While(..) {If(..){..}}

Statement	I If |S Switch |W While	Statement datatype is defined for three types of statement, which has three value constructors: If,Switch and While.
Switch	Switch String [Case]	Switch datatype is defined for Switch statement, which has two fields, one is String and another is the array of Case. As a result, Switch support the syntax like: Switch (state){Case 1:.Break Case 2:..Break Case ..:..}
Case	Case String [Expr]	Case datatype is defined for Case statement, which has two fields, one is string and another is the array of Expr
Program	E Expr|M Statement	Program datatype is defined for all program. There are two value constructors Expr and Statement. As a result, every line of program , except bracket, belongs to Expr or Statement.

5.3 Tokenization

In this section, the details about how to combine each parser to be parser combinator is described. As mentioned before, high order combinator assembly together to get a complicated combinator. The top parser combinator of all parser combinators is programP. The input file is string stream which flow to the top parser combinator programP. The output of programP is a binary tree, the left is error message during the parsing, and the right is token tree.

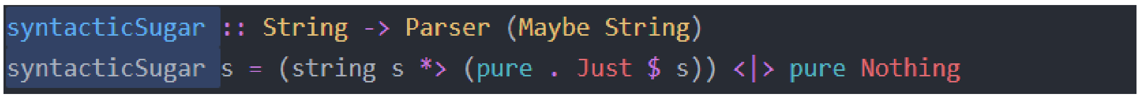
Figure 5.4 The structure of parser combinators tree



There are four kinds of parser.

- **The first one is for formatting the syntax.** For example, syntacitcSugar is used to ensure that even the string we expected does not exist, the program still can carry on parsing, so it is signification to secure the parser.

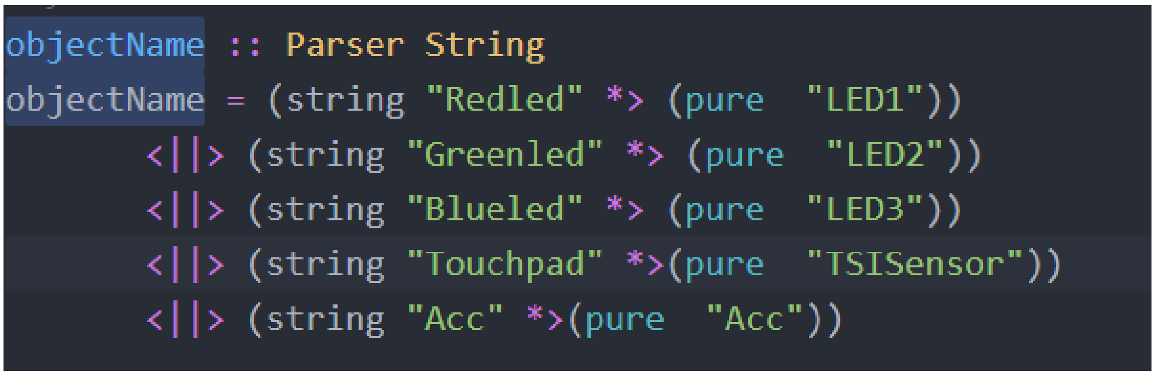
Figure 5.5 The code of syntacitcSugar[14]



```haskell
syntacticSugar :: String -> Parser (Maybe String)
syntacticSugar s = (string s *> (pure . Just $ s)) <|> pure Nothing
```

- **The second one is for parsing the code of ZYX to code of C++.**. For example, objectName.

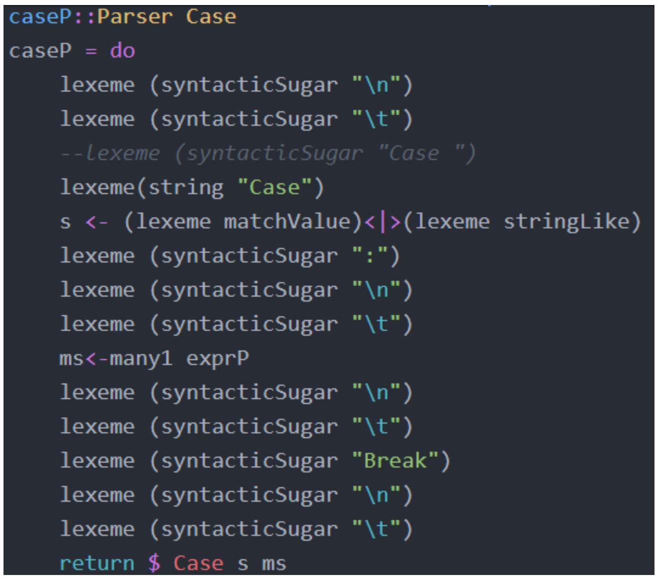
Figure 5.6 The code of objectName



```haskell
objectName :: Parser String
objectName = (string "Redled" *> (pure  "LED1"))
       <||> (string "Greenled" *> (pure  "LED2"))
       <||> (string "Blueled" *> (pure  "LED3"))
       <||> (string "Touchpad" *>(pure  "TSISensor"))
       <||> (string "Acc" *>(pure  "Acc"))
```

- **The third one** is like caseP for parsing these statements according to syntax.

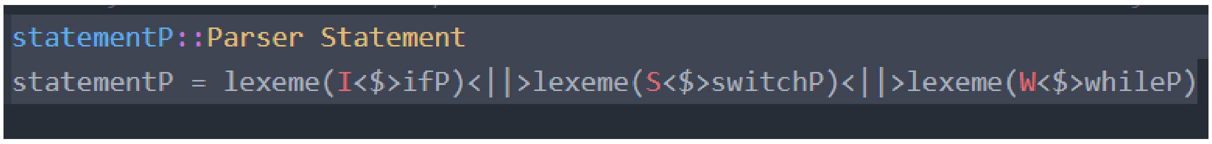
Figure 5.7 The code of caseP



```haskell
caseP::Parser Case
caseP = do
    lexeme (syntacticSugar "\n")
    lexeme (syntacticSugar "\t")
    --lexeme (syntacticSugar "Case ")
    lexeme(string "Case")
    s <- (lexeme matchValue)<|>(lexeme stringLike)
    lexeme (syntacticSugar ":")
    lexeme (syntacticSugar "\n")
    lexeme (syntacticSugar "\t")
    ms<-many1 exprP
    lexeme (syntacticSugar "\n")
    lexeme (syntacticSugar "\t")
    lexeme (syntacticSugar "Break")
    lexeme (syntacticSugar "\n")
    lexeme (syntacticSugar "\t")
    return $ Case s ms
```

- **The fourth one** is constructed by other parsers and contribute to its' parent parser. For example, statementP.

Figure 5.8 The code of statementP



```
statementP::Parser Statement
statementP = lexeme(I<$>ifP)<||>lexeme(S<$>switchP)<||>lexeme(W<$>whileP)
```

Figure 5.9 shows every explanation of parser. To make it easier to understand these functions , test sample is given for each function.

Figure 5.9 The explanation of each parser combinator

Parser name	Explanation
programP	parse the program be the array of Program For example: **input:** parse programP "" "greenled= Greenled \nredled= Redled\nWhile(True)\n{\n \n greenled =On\n blueled=Off\n}\n" **output:** Right [E (Exprobj {name = "greenled", operation = "=", value = "LED2"}),E (Exprobj {name = "redled", operation = "=", value = "LED1"}),M (W (While (Exprbool True) [M (S (Switch "" [Case "greenled " [Exprvar {name = "", operation = "=", value = "0"},Exprvar {name = "blueled", operation = "=", value = "1"}]]))]))]
Program	parse one line of program be one value constructor of self-defined datatype "Program" For example: **Input**: parse program "" "greenled= Greenled"

	Output: Right (E (Exprobj {name = "greenled", operation = "=", value = "LED2"}))
Statement	parse statement be one value constructor of self-defined datatype "Statement" For example: **Input**: parse statementP "" "While (True)\n{\n a=b\nc=d\n}" **Output**: Right (W (While (Exprbool True) [E (Exprvar {name = "a", operation = "=", value = "b"}),E (Exprvar {name = "c", operation = "=", value = "d"})])) **Input:** parse statementP "" "If(x > y)\n {x=y\n x=z\n}" **Output**: Right (I (If [IfE (Exprvar {name = "x ", operation = ">", value = "y"})] [Exprvar {name = "x", operation = "=", value = "y"},Exprvar {name = "x", operation = "=", value = "z"}]))
whileP	parse statement "While" to be self-defined datatype "While" For example: **input:** parse whileP "" "While (True)\n{\n a=b\nc=d\n}" **output:** Right (While (Exprbool True) [E (Exprvar {name = "a", operation = "=", value = "b"}), E (Exprvar {name = "c", operation = "=", value = "d"})])
Switch	parse statement "Switch" to be self-define datatype "Switch" For example: **input**: parse caseP "" "Case x: x=y\n x=z\n" **output**: Right (Case "x" [Exprvar {name = "x", operation = "=", value = "y"}, Exprvar {name = "x", operation = "=", value = "z"}]) lefe is error if there is mistake

caseP	parse statement "Case" to be self-define datatype "Case" for example: **input**: parse caseP "" "Case x: x=y\n x=z\n" **output**: Right (Case "x" [Exprvar {name = "x", operation = "=", value = "y"},Exprvar {name = "x", operation = "=", value = "z"}])
ifP	parse "If" statement to be self-define datatype If. For example **input**: parse ifP "" "If(x > y)\n {x=y\n x=z\n}" **output:** (If [IfE (Exprvar {name = "x ", operation = ">", value = "y"})] [Exprvar {name = "x", operation = "=", value = "y"},Exprvar {name = "x", operation = "=", value = "z"}])
ifconP	parse "if (condtion)" to be one value constructor of Ifcondtion datatype for example: **Input:** parse ifconP "" "x>y\n**"** **output:** Right (IfE (Exprvar {name = "x", operation = ">", value = "y"}))
exprP	parse the DSL expression to be one value constructor of Expr exprP= exprWait <||> exprBool <||> exprObject<||> exprValue
exprWait	parse the DSL "Wait" to self-datatype Exprwait For example: **input**: parse exprWait "" "Wait=0.2" **output**: Right (Exprwait "0.2")
exprBool	parse the DSL "True" or "False" to self-datatype Exprboo**l** **input:** parse exprBool "" "True" **output:** Right (Exprbool True)

exprValue	parse the DSL variable assignment to self-datatype Exprvar **input**: parse exprValue "" "a Eq b\n" **output:** Right (Exprvar {name = "a ", operation = "==", value = "b"})
exprObject	parse the DSL object assignment to self-datatype Exprobj For example: **Input:** parse exprObject "" "green=Greenled" **output:** Right (Exprobj {name = "green", operation = "=", value = "LED2"})
matchOpt	parse the DSL operation to C++ operation For example: **Input**: parse matchOpt "" "And" **Output**: Right "&&"
matchValue	parse the DSL reserve word to C++ object value For example: **Input:** parse matchValue "" "On" **Output:** Right "0"
objectName	parse the DSL object to mbed platform object name For example: **Input**: parse objectName "" "Redled" **Output:** Right "LED1"
syntacticSugar	syntacticSugar is used to ensure that even the string we expected is not exist, the program still can carry on parsing, so it is signification to secure the safe of parsing. For example:

	Input: parse (lexeme (syntacticSugar s)) "" s **Output**: Right (Just s)
(<||>)	(<||>) which basically takes two parsers and try to match the first one on the target;if it fails it tries the second one.
stringLike	stringLike is to keep all the parse string from the range. “abcdefghijklmnopqrstuvwxyz0.123456789CDXYZS !” For example: **Input**: parse stringLike “” “AV” **Output:** Right "" **Input**: parse stringLike “” “abc” **Output:** Right "abc"
Ws	parse space from string. For example: **Input**: parse ws "" " fsdfs " **output:** right “ "
Lexeme	Lexeme is used for skiping " " in parsing line. For example: **Input**: parse lexeme "" " fsdfs ” **output:** right “fsdfs “
Wait	wait is for paresing "Wait" expression, which is calling Thread::wait() in the millisecond range. For example: **input**: parse wait "" "Wait=0.2" **output**: Right "0.2"
Bool	bool is for combining parser "True" or "False" to Boolent True or False.
boolFalse	Parse "False" be False. For example: **input**: parse boolFalse "" "False"

	output: right "False"
boolTrue	Parse "True" be True **input**: parse boolTrue "" "True" **output**: right "True"

5.4 Data type checking

Once the program is checked as correct by compiler and get an abstract syntax tree, checking the program correctness according to datatype will be done. A set of functions, which name is “is”+datatype, will implement this job

For example:

The function isExprObj is for checking the Exprobj datatype. Figure5.10 shows the relationship between data type and datatype checking function.

Figure 5.10 The one-to-one correspondence between data types and check data type functions

Data type	Datatype checking function
Exprbool	isExprBool
Exprstring	isExprString
Exprwait	isExprWait
Exprvar	isExprVar
Exprobj	isExprObj
If	isIf
Switch	isSwitch
While	isWhile

Ifcondition	isIfE isIfO
Program	isE isM

5.5 Interpretation

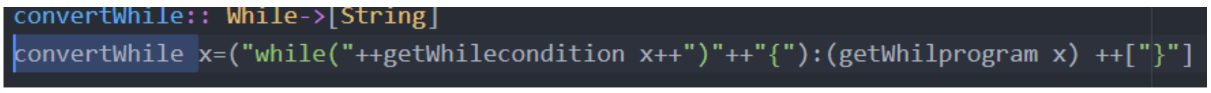
these tokens will be extracted sequentially from abstract syntax tree and validated. The validated tokens are transformed and formatted, then appending together. A group of functions will do extracting job, which name is defined as “get” + (the name of datatype) + (the name of value), for example: the function “getvarname” is used to get the name of “Exprvar” datatype.

Another important job in the stage is converting these data to be C++/C code. Several techniques are used for interpretation.

- **dynamic adding the reserve word of C++.**

these kinds of function’s name will start with “convert”. For example, “convertWhile” is for converting while statement from DSL code to C++ code.

Figure 5.6 The code of convertWhile

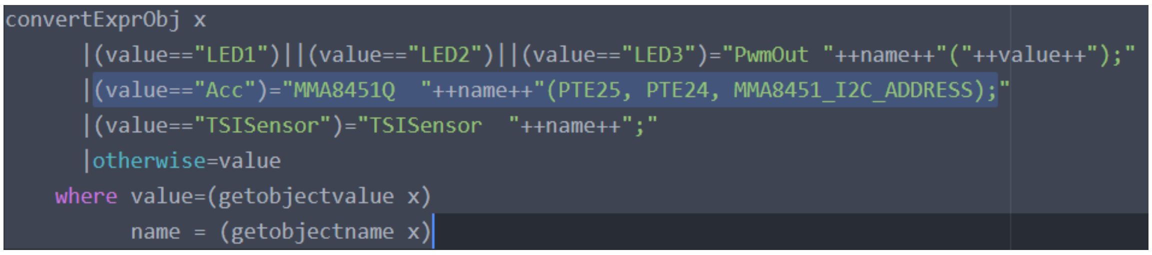


```
convertWhile:: While->[String]
convertWhile x=("while("++getWhilecondition x++")"++"{"):(getWhilprogram x) ++["}"]
```

- **dynamic replace the reserve word of ZYX to the reserve word of C++/C.**

For example “convertExprObj” is for replacing the definition of component of board from ZYX code to C++ code.

Figure 5.7 The code of convertExprObj

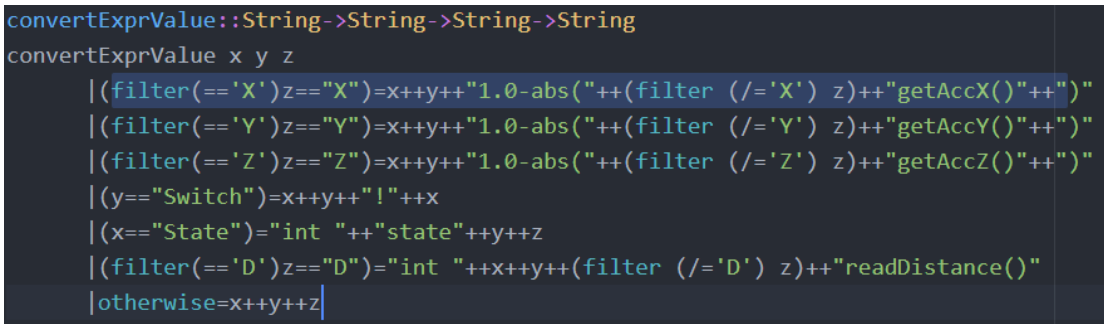


```
convertExprObj x
      |(value=="LED1")||(value=="LED2")||(value=="LED3")="PwmOut "++name++"("++value++");"
      |(value=="Acc")="MMA8451Q  "++name++"(PTE25, PTE24, MMA8451_I2C_ADDRESS);"
      |(value=="TSISensor")="TSISensor  "++name++";"
      |otherwise=value
    where value=(getobjectvalue x)
          name = (getobjectname x)
```

- **dynamic replace the function of ZYX to the function of Mbed SDK**

  For example, “convertExprValue” is for replacing the function of ZYX as the function of Mbed SDK.

Figure 5.8 The code of convertExprValue

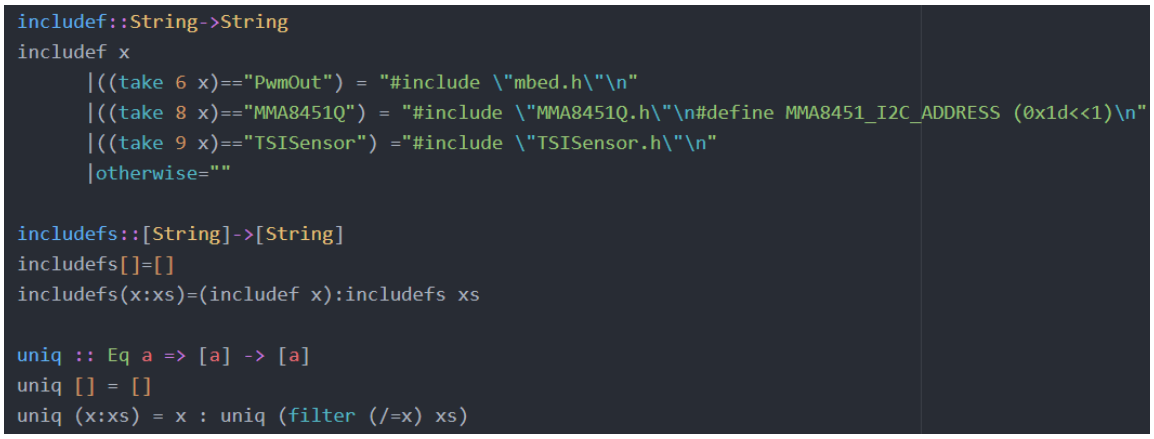


```
convertExprValue::String->String->String->String
convertExprValue x y z
      |(filter(=='X')z=="X")=x++y++"1.0-abs("++(filter (/='X') z)++"getAccX()"++")"
      |(filter(=='Y')z=="Y")=x++y++"1.0-abs("++(filter (/='Y') z)++"getAccY()"++")"
      |(filter(=='Z')z=="Z")=x++y++"1.0-abs("++(filter (/='Z') z)++"getAccZ()"++")"
      |(y=="Switch")=x++y++"!"++x
      |(x=="State")="int "++"state"++y++z
      |(filter(=='D')z=="D")="int "++x++y++(filter (/='D') z)++"readDistance()"
      |otherwise=x++y++z
```

5.6 Code generation

C++/C code is generated through a serial of convert function. To drive the ARM hardware, the executable file need to dynamic add the component library base on the object end-user defined. For example, the function includefs checks to see if the code contains “Acc”, and if so, the final code needs to include MMA8451Q.h header file and needs define the address. uniq is used to remove duplicated header file. Figure 5.9 is the key Haskell code.

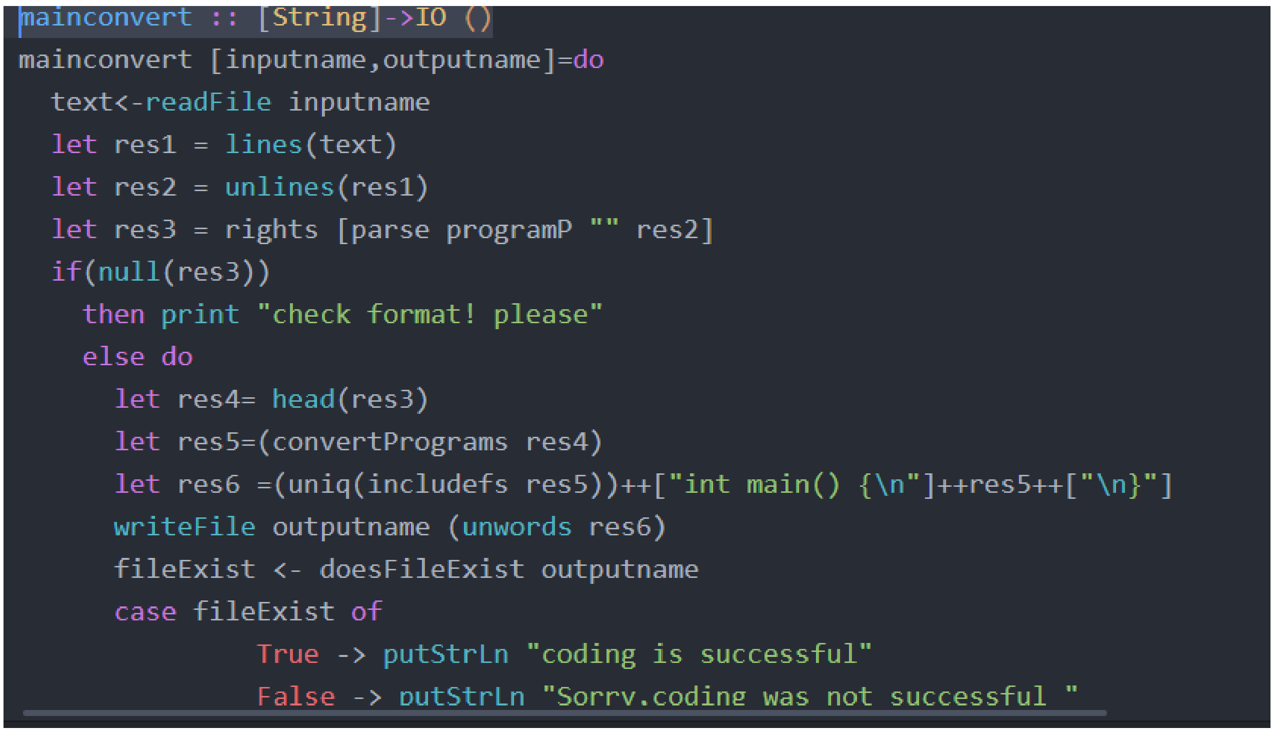
Figure 5.9 The code of ZYX includef,includefs and uniq



```
includef::String->String
includef x
      |((take 6 x)=="PwmOut") = "#include \"mbed.h\"\n"
      |((take 8 x)=="MMA8451Q") = "#include \"MMA8451Q.h\"\n#define MMA8451_I2C_ADDRESS (0x1d<<1)\n"
      |((take 9 x)=="TSISensor") ="#include \"TSISensor.h\"\n"
      |otherwise=""

includefs::[String]->[String]
includefs[]=[]
includefs(x:xs)=(includef x):includefs xs

uniq :: Eq a => [a] -> [a]
uniq [] = []
uniq (x:xs) = x : uniq (filter (/=x) xs)
```

When all the code is ready, an output file is written. The default output filename is “mbedout.c”. end-user can define any name for input and output file by adding two parameters to the execution compiler. The first parameter is for input file name, the second is for output filename. The function “mainconvert” assemble all function together and control the general flow to write C++/C file.

Figure 5.10 The code of mainconvert



```
mainconvert :: [String]->IO ()
mainconvert [inputname,outputname]=do
  text<-readFile inputname
  let res1 = lines(text)
  let res2 = unlines(res1)
  let res3 = rights [parse programP "" res2]
  if(null(res3))
    then print "check format! please"
    else do
      let res4= head(res3)
      let res5=(convertPrograms res4)
      let res6 =(uniq(includefs res5))++["int main() {\n"]++res5++["\n}"]
      writeFile outputname (unwords res6)
      fileExist <- doesFileExist outputname
      case fileExist of
               True -> putStrLn "coding is successful"
               False -> putStrLn "Sorry,coding was not successful "
```

6.Test

6.1 White box test

6.1.1 Statement and Branch Coverage test

During these tests, every line and branch of the code has been tested. The following is a sample of test case in the project.

Figure 6.1 The case of statement test

Function name	**Input**	**output**
boolTrue	parse boolTrue "" "True"	right "True"
boolFalse	parse boolFalse "" "False"	Right "False"
bool	parse bool "" "False”	Right “False””
	parse bool "" "True”	Right "True"
wait	parse wait "" "Wait=0.2"	Right "0.2"
ws	parse ws "" " fsdfs "	Right " "
lexeme	parse lexeme "" " fsdfs "	Right " fsdfs "
stringLike	parse stringLike "" "sfssf"	Right "sfssf"
	parse stringLike "" "SF"	Right "S"
	parse stringLike "" "ABC"	Right ""
syntacticSugar	parse ((syntacticSugar "sfsd")) "" "sfsd"	Right (Just "sfsd")
	parse ((syntacticSugar "sfsd")) "" "fsd"	Right Nothing
objectName	parse objectName "" "Redled"	Right "LED1"
	parse objectName"" "Greenled"	Right "LED2"

	parse objectName"" "Blueled"	Right "LED3"
	parse objectName"" "Acc"	Right "Acc"
	parse objectName "" "Touchsensor"	Right "TSISensor"
matchValue	parse matchValue "" "On"	Right "0"
	parse matchValue "" "Off"	Right "1"

6.1.2 Unit coverage test

Except testing every line and branch in the code, according to the module architecture and ZYX process flow, five main unit test have done.

Figure 6.2 The flow chart of unit coverage test

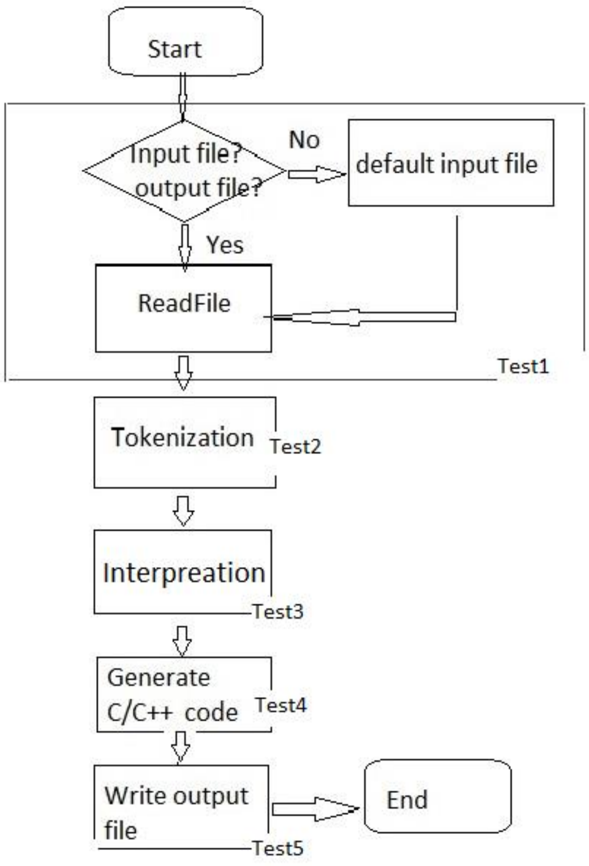


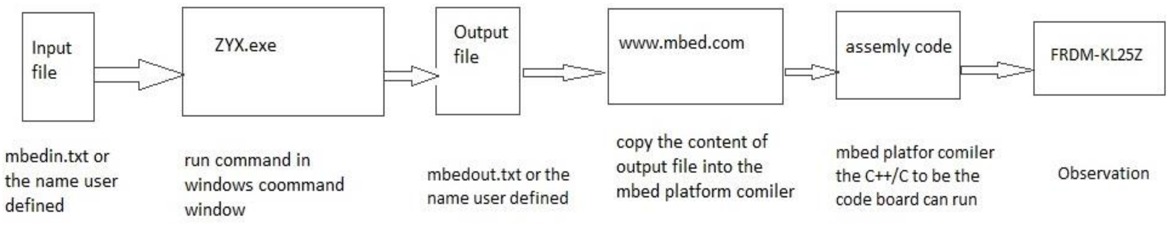
Figure 6.3 The test case of unit coverage test

Test name	Input	Output
Test 1: Input file	mbedin.txt	"greenled= Greenled \nredled= Redled\nWhile(True)\n{\n \n greenled =On\n blueled=Off\n}\n"
Test 2: Tokenization	The output of test1	Right [E (Exprobj {name = "greenled", operation = "=", value = "LED2"}),E (Exprobj {name = "redled", operation = "=", value = "LED1"}),M (W (While (Exprbool True) [M (S (Switch "" [Case "greenled " [Exprvar {name = "", operation = "=", value = "0"},Exprvar {name = "blueled", operation = "=", value = "1"}]]))]))]
Test 3: Interpretation	The output of test2	["PwmOut greenled(LED2);\n","PwmOut redled(LED1);\n","while(true){\nswitch()\n{\ncase greenled :\n=0;\nblueled=1;\nbreak;\n\n}\n\n}\n"]
Test 4: Generate C++/C code	The output of test3	["#include \"mbed.h\"\n","","int main() {\n","PwmOut greenled(LED2);\n","PwmOut redled(LED1);\n","while(true){\nswitch()\n{\ncase greenled :\n=0;\nblueled=1;\nbreak;\n\n}\n\n}\n","\n}"]
Test 5: Write output file	The output of Test5	mbedout.c

6.1.3 Black box test

The Black box asked end-user for helping test. End-user only knew testing process, who didn’t know the implementation details. The Figure 6.4 shows the testing process.

Figure 6.4 The test case of black box test


Input file
ZYX.exe
Output file
www.mbed.com
assemly code
FRDM-KL25Z
mbedin.txt or the name user defined
run command in windows coommand window
mbedout.txt or the name user defined
copy the content of output file into the mbed platform comiler
mbed platfor comiler the C++/C to be the code board can run
Observation

There are four black test cases, which tried to cover every line and branch in the code.

Case 1: White colour light on board will blink three times. the interval time is 0.2 second

DSL source code	C++ source code
redled= Redled	#include "mbed.h"
blueled= Blueled	int main() {
greenled= Greenled	PwmOut redled(LED1);
int repeat=3	PwmOut blueled(LED3);
While(repeat>0)	PwmOut greenled(LED2);
{	int repeat=3 ;
redled = On	while(repeat>0){
blueled = On	redled =0;
greenled= On	blueled =0;

Wait=0.2
 	
redled = Off
 	
blueled = Off
 	
greenled= Off
 	
Wait=0.2
 	
repeat=repeat-1
 	
}
 	
}
	greenled=0;
	
	wait(0.2);
	
	redled =1;
	
	blueled =1;
	
	greenled=1;
	
	wait(0.2);
	
	repeat=repeat-1;
	
	}
	
	}

Case 2: The three LED flash in turns. The sequence is Red->Green ->Blue, then back tor Red->Green ->Blue for loop.

DSL source code	C++ source code

greenled= Greenled
 	
redled= Redled
 	
blueled= Blueled
 	
int state= 1
 	
While(True)
 	
{
 	
Wait=1
 	
Switch(state)
 	
{
 	
Case 1:
 	
redled=On
	#include "mbed.h"
	
	int main() {
	
	PwmOut greenled(LED2);
	
	PwmOut redled(LED1);
	
	PwmOut blueled(LED3);
	
	int state=1;
	
	while(true){
	
	wait(1);
	
	switch(state)
	
	{

```
            blueled=Off
           greenled=Off
            state =2
            Break
      Case 2
            blueled=On
  redled=Off
     greenled=Off
            state =3
            Break
        Case 3
             greenled=On
 redled=Off
   blueled=Off
             state=1
             Break
 }
 }
```
	```
	                case 1:
	                redled=0;
	                blueled=1;
	                greenled=1;
	                state =2;
	                break;
	                case 2:
	                blueled=0;
	                redled=1;
	                greenled=1;
	                state =3;
	                break;
	                case 3:
	                greenled=0;
	                redled=1;
	                blueled=1;
	                state=1;
	                break;
	}}}
	```

Case 3: When moving the board to different position will lit different colour LED. Keeping the board is flat, then the LED colour is green; keeping the board vertical, then the LED colour is red. Moving the board up and down, the LED colour is blue.

DSL source code	**C++ source code**
acc=Acc	#include "mbed.h"

led1=Redled
 	
led2=Greenled
 	
led3=Blueled
 	
While(True)
 	
{
 	
    led1=acc.X
 	
    ed2=acc.Y
 	
    led3=acc.Z
 	
}
	#include "MMA8451Q.h"
	
	#define MMA8451_I2C_ADDRESS (0x1d<<1)
	
	int main(void) {
	
	   MMA8451Q acc(PTE25, PTE24, MMA8451_
	
	I2C_ADDRESS);
	
	   PwmOut rled(LED_RED);
	
	   PwmOut gled(LED_GREEN);
	
	   PwmOut bled(LED_BLUE);
	
	   while (true) {
	
	      rled = 1.0 - abs(acc.getAccX());
	
	      gled = 1.0 - abs(acc.getAccY());
	
	      bled = 1.0 - abs(acc.getAccZ());
	
	      wait(0.1);
	
	   }
	
	}

Case 4: Touching left part of touchpad led to Green LED on/off. Touching the middle part will switch on /off Red LED on and the right part will switch on/off Yellow LED on.

DSL source code	C++ source code

greenled= Greenled
 	
redled= Redled
 	
blueled= Blueled
 	
tsi= Touchpad
 	
While(True)
	#include "mbed.h"
	
	 #include "TSISensor.h"
	
	  int main() {
	
	 PwmOut greenled(LED2);
	
	 PwmOut redled(LED1);

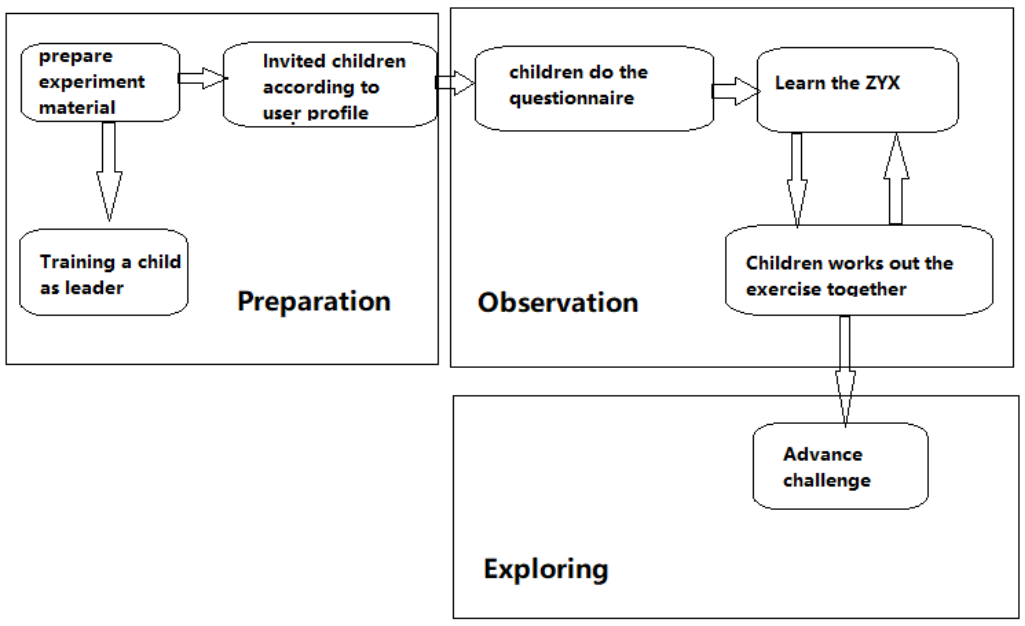
{	PwmOut blueled(LED3);
distance=tsi.D	TSISensor tsi;
If (distance>0 And	while(true){
distance<13)	int distance=tsi.readDistance());
{	if(distance>0 && distance <13)
redled =!redled	{
}	redled = !redled;
If (distance>13 And	}
distance<26)	if(distance>13 && distance <26)
{	{
greenled=!greenled	greenled =!greenled;
}	}
If (distance>26 And	if(distance>26 && distance <40)
distance<40)	{
{	blueled =!blueled ;
blueled=! blueled	}
}	}
}	}

7.Evaluation

7.1 Experiments general procedure

Evaluation is the most important stage to a new language. A group of children (9-11) is organised. The team has a leader who has be trained for the ZYX. Team leader who is 10 years old will lead other children to implement the programming task. All children will not know the programming task in advance. There are at least three benefits to the organization. First, when children perform task together, they will not give up easily. Second, it is easy to observe the level of understanding of team leader because the leader must solve other member's different problem. Third, learning to teach child in a child's way is a good reference to improve performance of ZYX. Figure 7.1 shows key tasks need to do during the experiment.

Figure 7.1 The procedure of experiments


prepare experiment material
Invited children according to user profile
Training a child as leader
Preparation
children do the questionnaire
Learn the ZYX
Children works out the exercise together
Observation
Advance challenge
Exploring

- **Unit one: Preparation**
  1. Prepare all experiment materials, includes questionnaire, introduction, exercise and measurement.
  2. Inviting children who match the user profile.
  3. Inviting a child as team leader to be trained.
- **Unit two: Observation**
  4. Giving questionnaire to children.

5. Organizing group, introduce ZYX.
6. Explaining the programming task and work out the task as group.
7. Recording some data for measurement, for example: how many tasks finished, how long each task taken, how many times ask helper and so on.

- **Unit three: Exploring**

8. Do advanced exercise for evaluating the usability of ZYX and the ability of children using the ZYX,

7.2 Materials for the experiments

- **Questionnaire**

This paper is for knowing the level of computer science and programming in this age

- **Slides**

All the concept of sample code about the DSL are included in the slides, which could help each child consult his question easily and self-study. After finishing the group task, children will do some exercises for evaluating the children’s ability to create one complex behaviour of ARM board.

- **Measurement sheet**

During the experiment, using a measurement sheet to record some data for identifying the children’s learning curve of ZYX and the depth of children learning.

7.3 Equipment and environment of experiment

The same working equipment and stable environment are very important to experiment. The working environment prepared for the user can also influent result. The working environment includes software, hardware and place.

Figure 7.2 the equipment and environment of experiment

Software:	Operating System: Microsoft windows 10 Home Editor: Notepad Compiler: ZYX.exe
ARM board:	FRDM-KL25Z
Laptop (HP 14):	Processor: Intel Core i3-7020U@2.30GHZ RAM:4GB
Place	Home with biscuits

7.4 Feedback

- **The understanding of concept**

Children can quickly understand the difference of variable and object name. After practising several times, children will notice the lowercase and Capitan without any mistake. Statements and algorithms are easy understand by children except “switch case”.

- **The time spending**

Simple task, such as switching on two LEDs, children can write the code in one minute with no reference sample and no errors. To moderate level job, for example, use “while” to control LED flash, the average time is five minutes with no reference sample. The hard job, children can read case sample to complete it. The helper needs to remind some mistakes. All the time is less 15 minutes.

- **The common errors**

Forget the object name need to be capitalized and like messy the format of variable are common syntax errors. Wait is a tricky function for helping the LED flash, Sometimes, need remind to put the right place.

- **Comments**

Children's ability to learn at ZYX is impressive. The language did not be explained. After several cases are demonstrating, children are so excited to try the different algorithm. Children all said ZYX is an easy and interesting language to master especially compare the C++/C code that convert by ZYX compiler.

8. Discussion and Conclusions

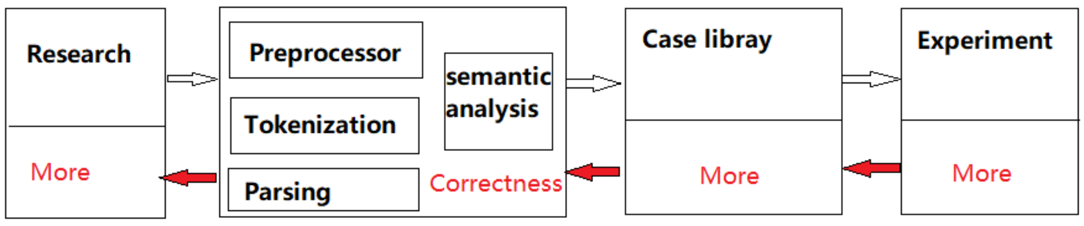
Through a set of experiments, three main findings are discovered. First one, textual programming language is not a barrier to children, the barrier is the effectiveness and complexity of program. This is not only a child's barrier, but also an adult's learning issue as well. Second one, ZYX keeps general structure of programming language, which is really help children to understand general programming language. For example, C++/C code can be guessed and read after mastering ZYX. The third one, using DSL for embedded system is a right way for drawing children interesting and encourage them self-study. The result is in agreement with previous analysis, designing a suitable textual programming language for children in some domain will greatly improve their ability to learn programming skills and computational science knowledge.

Without doubt, because limitation of time, so loads of job about the implementation and experiment of ZYX need to be done in future. For example, the sematic verify is very poor for ZYX currently. the number of users in experiment is small. The comparison with other graphic programming language is not involved.

9.Further work

The project has gone through a path from research to practice, but this experience is still not enough to establish a theoretical system about how to design a textual programming language for children. So at least two tasks need to be done in future. First one is improving the capability and usability of ZYX.

Figure 9.1 the process of how to improve ZYX's performance



- doing more experiments and inviting more children to join testing ZYX.
- Building more Case for ZYX
- Improving correctness
- Do more research to perfect ZYX

The second one is comparing ZYX and graphical programming language. The best option is the BBC micro-bit, which has a similar embedded board the project. Because the BBC micro-bit supports Python, so If another compiler, which can convert ZYX code to Python code, be created. Some interesting experiments might be done. For example, a group child who learning graphical programming language in micro-bit and another group child who learning ZYX, and both using the same time. Then teaching Python together and doing some Python exercise, which group is better? I am really looking forward to seeing it!

10.Reference

10.1 Embedded platform

[1] Kothari, D.P. & ebrary, I. 2012;2011;, Embedded systems, New Age International, New Delhi.

[2] oulson, R., Wilmshurst, T. & Books24x7, I. 2012, Fast and effective embedded systems design: applying the ARM mbed, 1st edn, Elsevier, Boston, MA;Oxford;

10.2 Children programming languages and tools

[3]Rogers, Y., Shum, V., Marquardt, N., Lechelt, S., Johnson, R., Baker, H. & Davies, M. 2017, "From the BBC micro to micro:bit and beyond: a British innovation", interactions, [Online], vol. 24, no. 2, pp. 74-77

[4] Tsukamoto, H., Takemura, Y., Nagumo, H., Ikeda, I., Monden, A. & Matsumoto, K. 2015, "Programming education for primary school children using a textual programming language", IEEE, , pp. 1.

[5] The Lifelong Kindergarten Group at the MIT Media Lab n.d, The interface of Scratch. Available from:https://scratch.mit.edu/projects/editor/?tutorial=getStarted. [13 October 2019].

[6] Google n.d, The guides of Blockly,Available from:https://developers.google.com/blockly/guides/overview. [14 October 2019].

[7] BBC n.d, The guides of micro:bit,Available from:https://microbit.org/guide/. [10 October 2019].

10.3 Research papers and books

[8] Bach, C., Miernik, A. & Schönthaler, M. 2014, "Training in robotics: The learning curve and contemporary concepts in training", Arab Journal of Urology, vol. 12, no. 1, pp. 58-61.

[9] Department for Education in UK 09.2013, National curriculum in England: computing programmes of study, Available

from:https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study. [14 June,2019].

[10] Martin Fowler 2011, Domain Specific Languages, Available from: https://martinfowler.com/books/dsl.html. [14 January,2019].

[11] Frost, R. & Hafiz, R. 2006, "A new top-down parsing algorithm to accommodate ambiguity and left recursion in polynomial time", ACM SIGPLAN Notices, vol. 41, no. 5, pp. 46-54.

[12] Bell, P. 2007, "A practical high volume software product line", ACM, , pp. 994.

10.4 Programming language

[13] O'Sullivan, B., Stewart, D. & Goerzen, J. 2009, Real world Haskell, 1st edn, O'Reilly, Sebastopol, CA.

[14] Alfredo Di Napoli 07.2013, Episode 5 - A simple DSL,Available from:https://www.schoolofhaskell.com/user/adinapoli/the-pragmatic-haskeller/episode-5-a-simple-dsl. [13 June 2019].

10.5 Tools

[15] ARM n.d, An introduction to Arm Mbed OS 5,Available from:https://os.mbed.com/docs/mbed-os/v5.13/introduction/index.html. [13 May 2018].

A. Appendix

A.1 Questionnaire

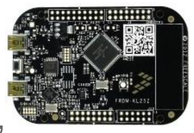
ZYX Survey

I have designed a new programming language ZXY to a pocket-sized hobby computer for helping children to build their basic knowledge about computer science, especially in programming. This survey is helping me to understand the speed and ways of learning in 9-11 years old children.

1.What is a computer? The following lists you think it is not a computer, please cross it.

Laptop tablets smartphone

micro-computer which look like:


,

2.What is hardware in computer?

For example,

3.What is software in computer?

For example,

4.What is a bug in computer?

5.What is the brain of computer?

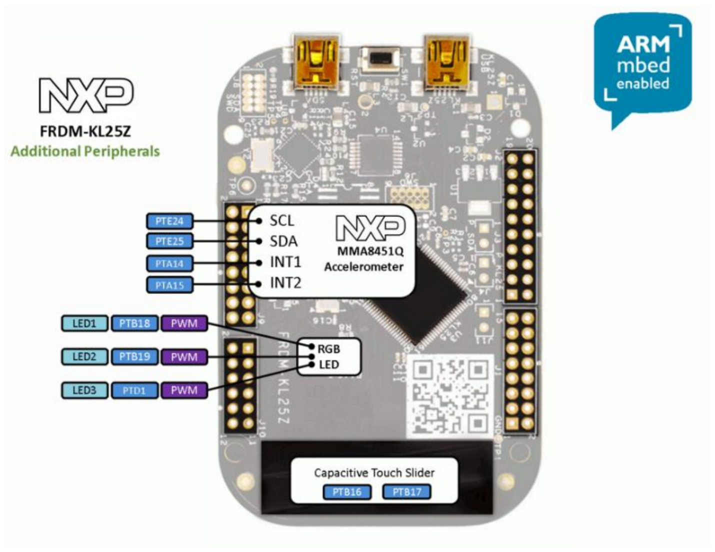
Computer chips is the brain of computer, the following do you think which one is the "chips"?

6. What is the three basic colours of screen?

The screen on computer allows the computer to show its results, which is a lit-up panel covered with two layers of filters. One filter controls the brightness and the other controls the colour. Touchscreens have an extra layer that can sense tiny changes. All the colours on the screen are made by mixing just three colours of light, guess which three colours of light?

7.Your first program using ZYX

This is a tiny, built -in computers, we can find inside everything from cars to cameras. Then we will have a programming language the name is ZYX to do some fab works. There are three LED light just like computer screen.



We can program to change the different colour: Redled is for the red LED, Blueled is for the blue LED, Yellowled is for the yellow LED.

Sample1: I have defined a name for my led light and set it is colour

myfirstled=Redled

myfirstled is the name you define for your led, just notice all are lowercase

Program 1: Could you define a name for your LED?

Program 2: Could you set your LED as blue LED or yellow LED?

9.More programs using ZYX

Sample1:switch off LED.

myfirstled=Off

Sample2: switch on LED

myfirstled=On

Sample 3: Switch on The Red LED, The Green LED and Blue LED to mix the white colour of LEDs.

```
myled1=Redled

myled2=Blueled

myled3 =Yellowled

myled1=On

myled2=On

myled3=On
```

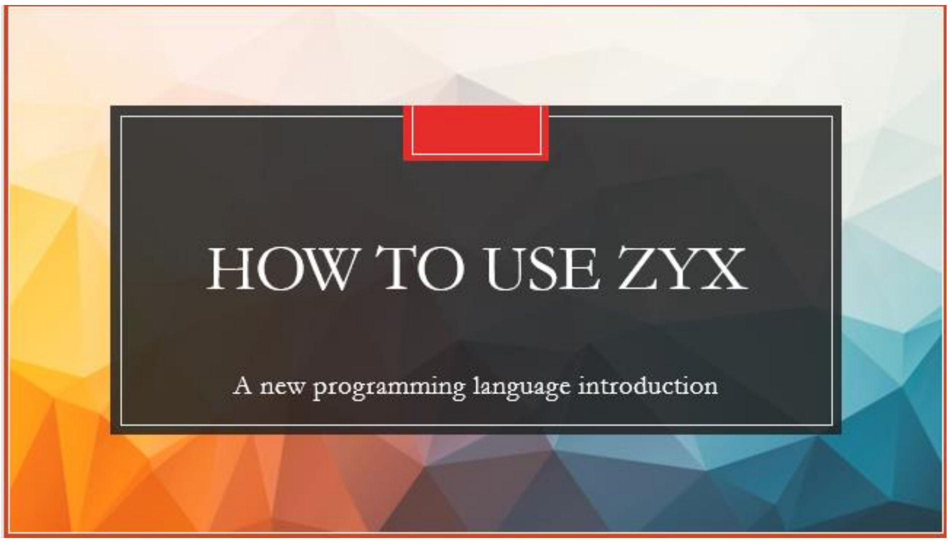
Could you write a program let the LEDs change to purple colour?

10. Could you easy to read and work out what will happen?

```
myled= Greenled

While(True)

{

    myled =  On

    Wait=0.2

    myled=Off

   Wait=0.2

}
```

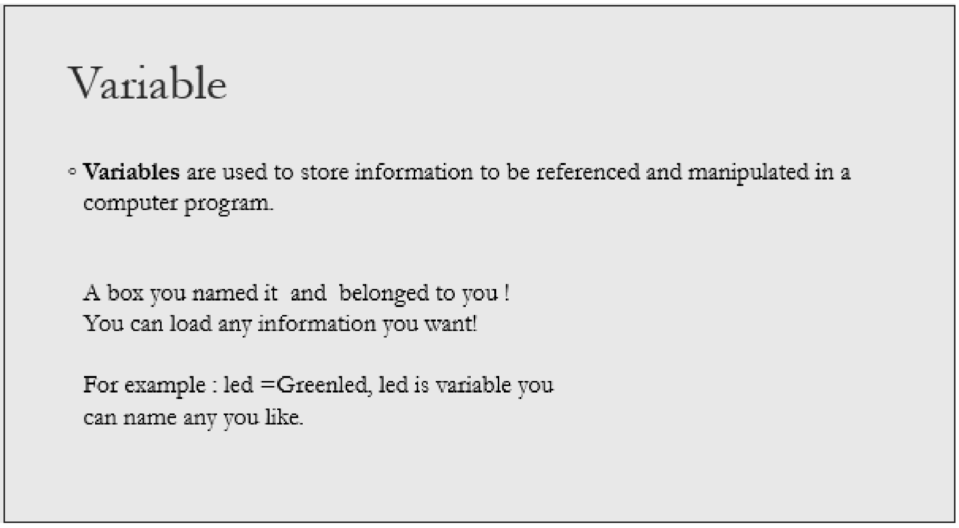
Thank you so much for your supporting! Are you interested in it and would like to try out a new programming language ZYX to observe how it works?

A.2 Slides



HOW TO USE ZYX

A new programming language introduction

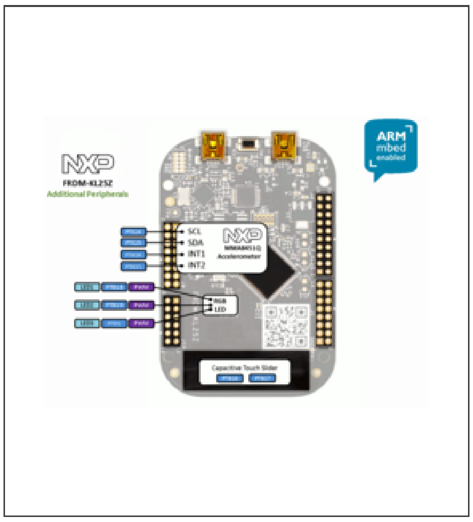


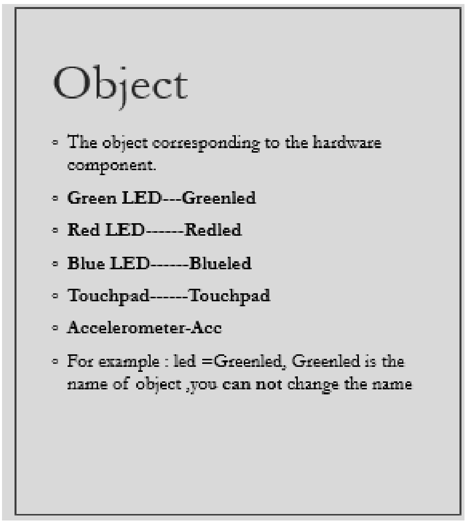
Variable

- **Variables** are used to store information to be referenced and manipulated in a computer program.

A box you named it and belonged to you !
You can load any information you want!

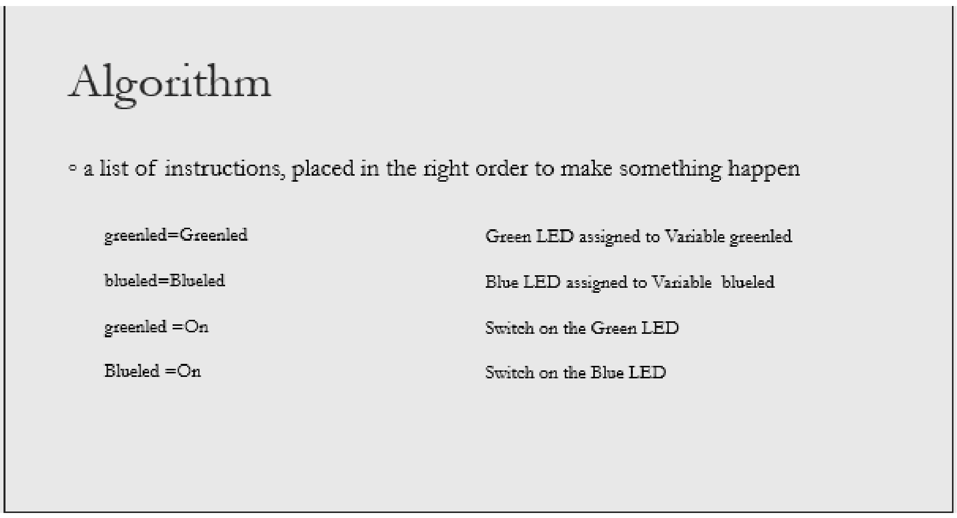
For example : led =Greenled, led is variable you can name any you like.





Object

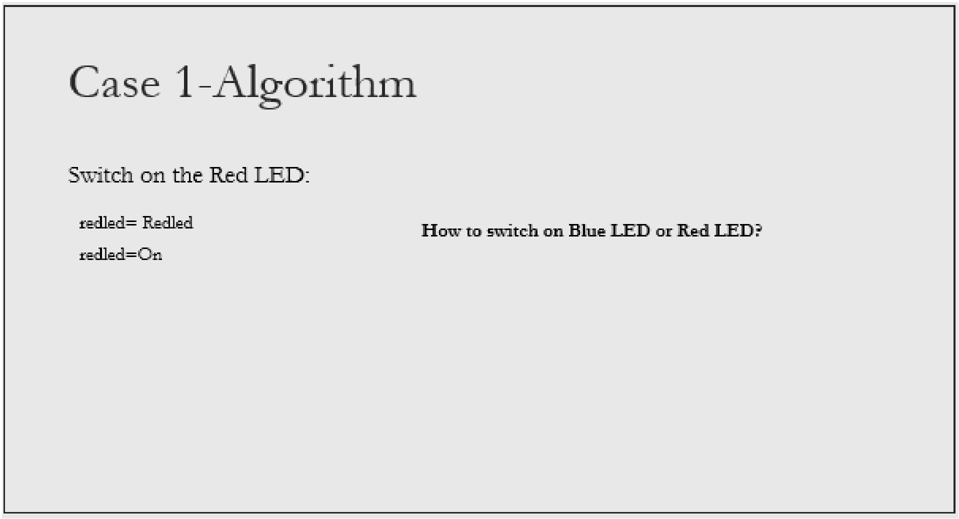
- The object corresponding to the hardware component.
- **Green LED---Greenled**
- **Red LED------Redled**
- **Blue LED------Blueled**
- **Touchpad------Touchpad**
- **Accelerometer-Acc**
- For example : led =Greenled, Greenled is the name of object ,you can not change the name



Algorithm

- a list of instructions, placed in the right order to make something happen

greenled=Greenled	Green LED assigned to Variable greenled
blueled=Blueled	Blue LED assigned to Variable blueled
greenled =On	Switch on the Green LED
Blueled =On	Switch on the Blue LED

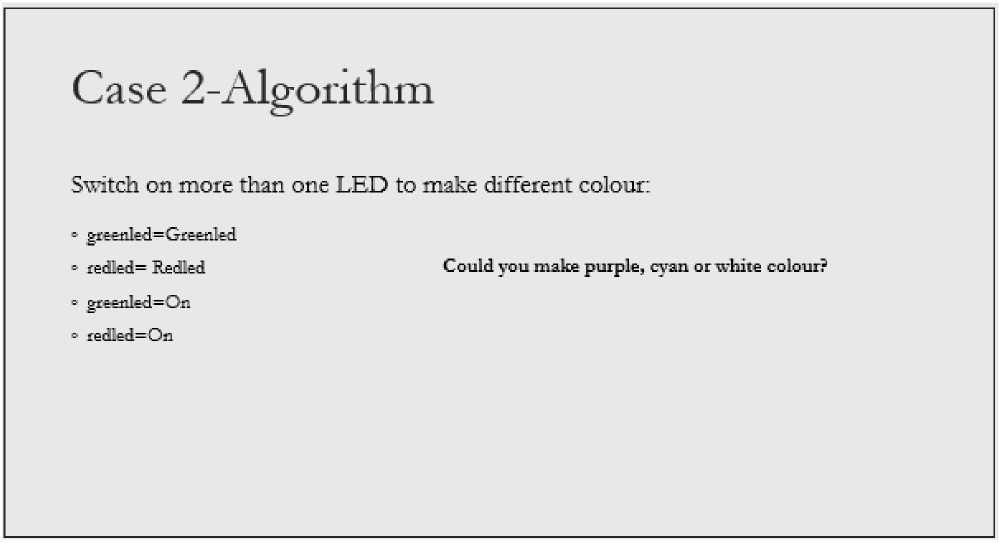


Case 1-Algorithm

Switch on the Red LED:

redled= Redled
redled=On

How to switch on Blue LED or Red LED?

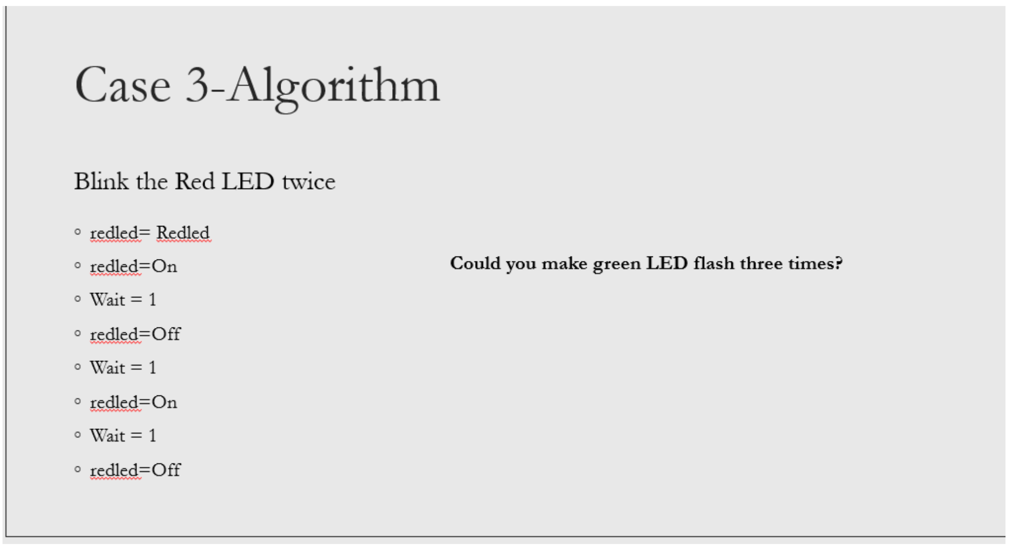


Case 2-Algorithm

Switch on more than one LED to make different colour:

- greenled=Greenled
- redled= Redled
- greenled=On
- redled=On

Could you make purple, cyan or white colour?

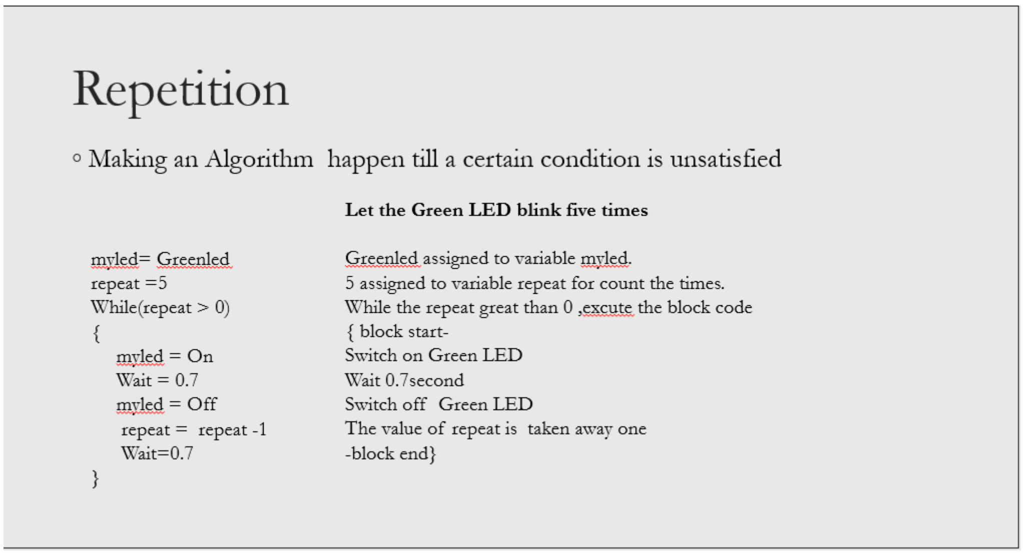


Case 3-Algorithm

Blink the Red LED twice

- redled= Redled
- redled=On
- Wait = 1
- redled=Off
- Wait = 1
- redled=On
- Wait = 1
- redled=Off

Could you make green LED flash three times?

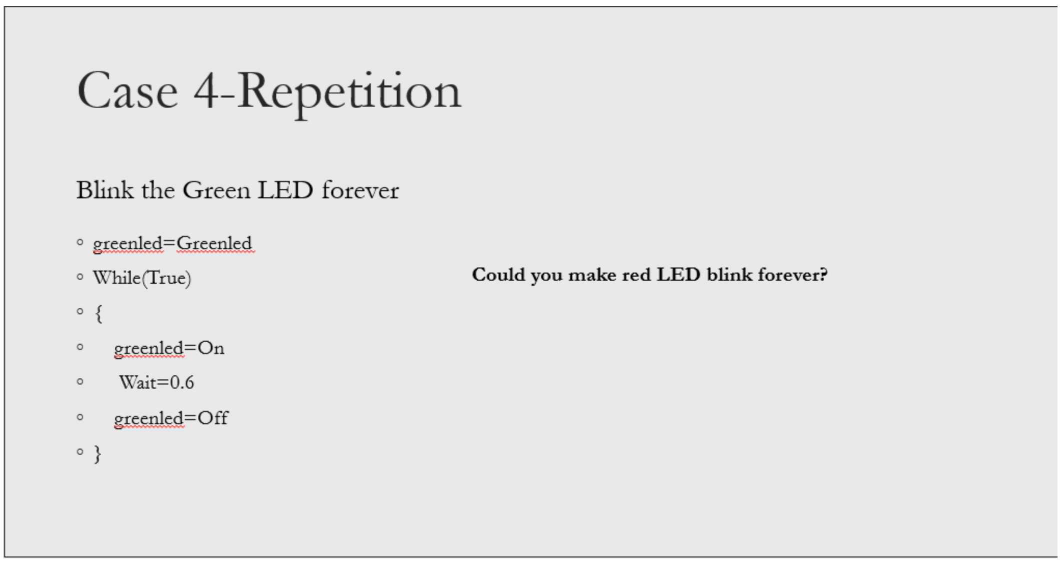


Repetition

- Making an Algorithm happen till a certain condition is unsatisfied

Let the Green LED blink five times

```
myled= Greenled            Greenled assigned to variable myled.
repeat =5                  5 assigned to variable repeat for count the times.
While(repeat > 0)          While the repeat great than 0 ,excute the block code
{                          { block start-
   myled = On              Switch on Green LED
   Wait = 0.7              Wait 0.7second
   myled = Off             Switch off  Green LED
    repeat =  repeat -1    The value of repeat is  taken away one
    Wait=0.7               -block end}
}
```

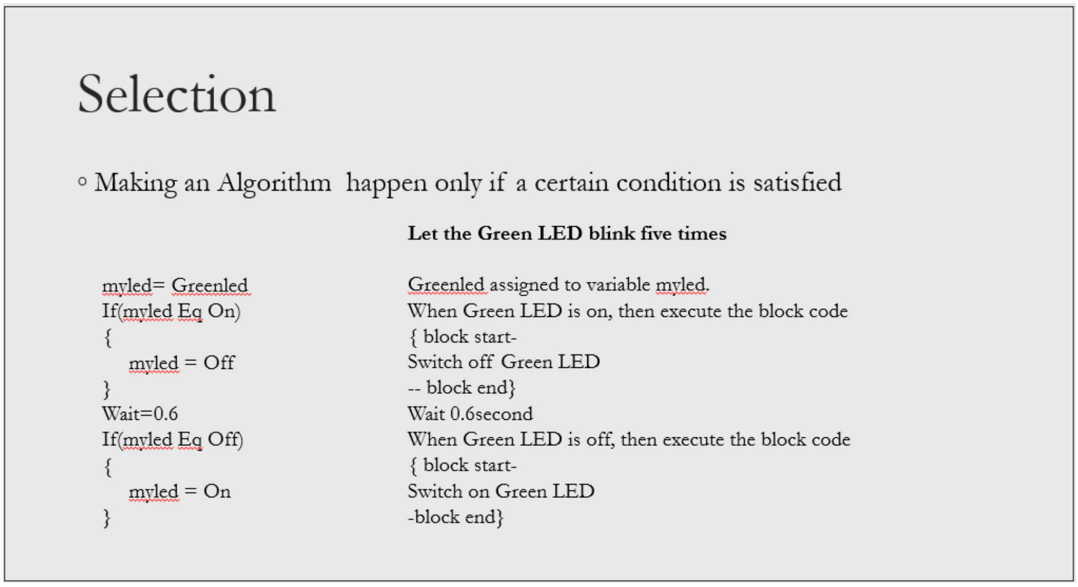


Case 4-Repetition

Blink the Green LED forever

```
◦ greenled=Greenled
◦ While(True)
◦ {
◦    greenled=On
◦     Wait=0.6
◦    greenled=Off
◦ }
```

Could you make red LED blink forever?

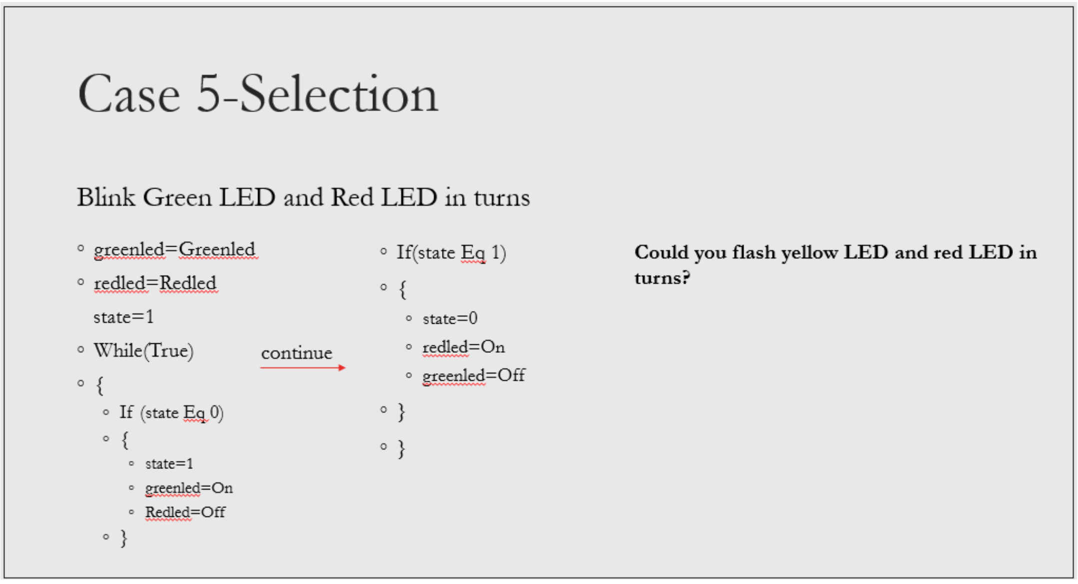


Selection

- Making an Algorithm happen only if a certain condition is satisfied

Let the Green LED blink five times

```
myled= Greenled          Greenled assigned to variable myled.
If(myled Eq On)          When Green LED is on, then execute the block code
{                        { block start-
   myled = Off           Switch off Green LED
}                        -- block end}
Wait=0.6                 Wait 0.6second
If(myled Eq Off)         When Green LED is off, then execute the block code
{                        { block start-
   myled = On            Switch on Green LED
}                        -block end}
```

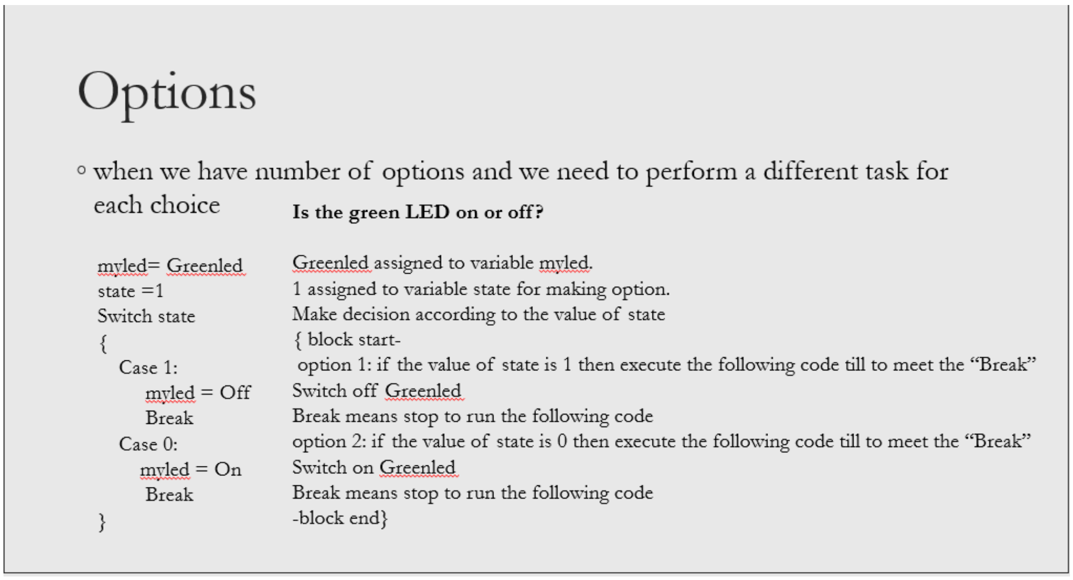


Case 5-Selection

Blink Green LED and Red LED in turns

```
◦ greenled=Greenled              ◦ If(state Eq 1)
◦ redled=Redled                  ◦ {
  state=1                            ◦ state=0
◦ While(True)      continue →        ◦ redled=On
◦ {                                  ◦ greenled=Off
   ◦ If (state Eq 0)             ◦ }
   ◦ {                           ◦ }
      ◦ state=1
      ◦ greenled=On
      ◦ Redled=Off
   ◦ }
```

Could you flash yellow LED and red LED in turns?

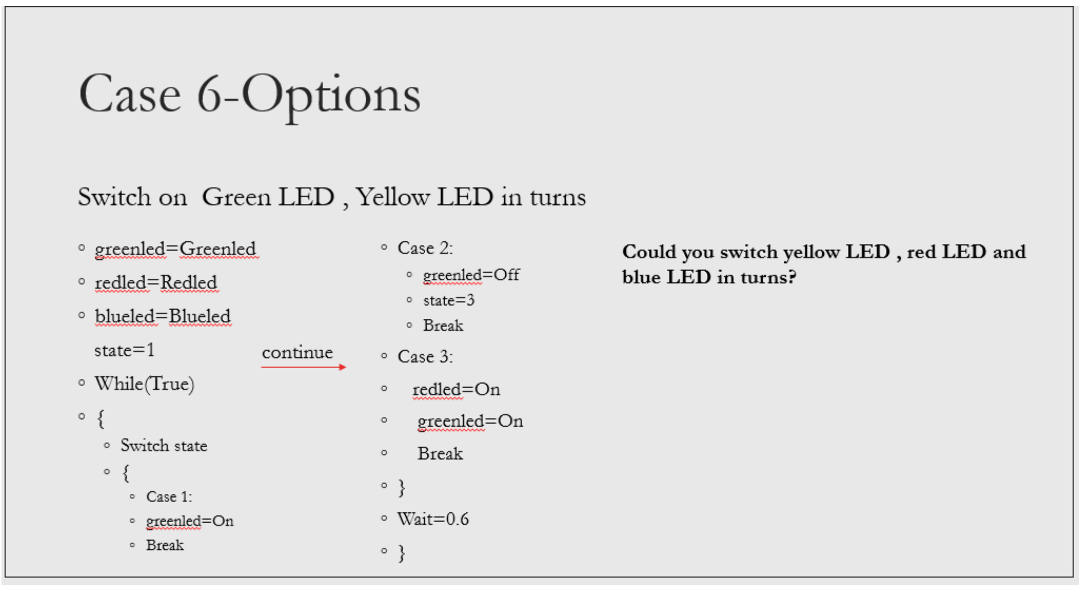


Options

- when we have number of options and we need to perform a different task for each choice

Is the green LED on or off?

```
myled= Greenled          Greenled assigned to variable myled.
state =1                 1 assigned to variable state for making option.
Switch state             Make decision according to the value of state
{                        { block start-
   Case 1:               option 1: if the value of state is 1 then execute the following code till to meet the "Break"
      myled = Off        Switch off Greenled
      Break              Break means stop to run the following code
   Case 0:               option 2: if the value of state is 0 then execute the following code till to meet the "Break"
      myled = On         Switch on Greenled
      Break              Break means stop to run the following code
}                        -block end}
```



Case 6-Options

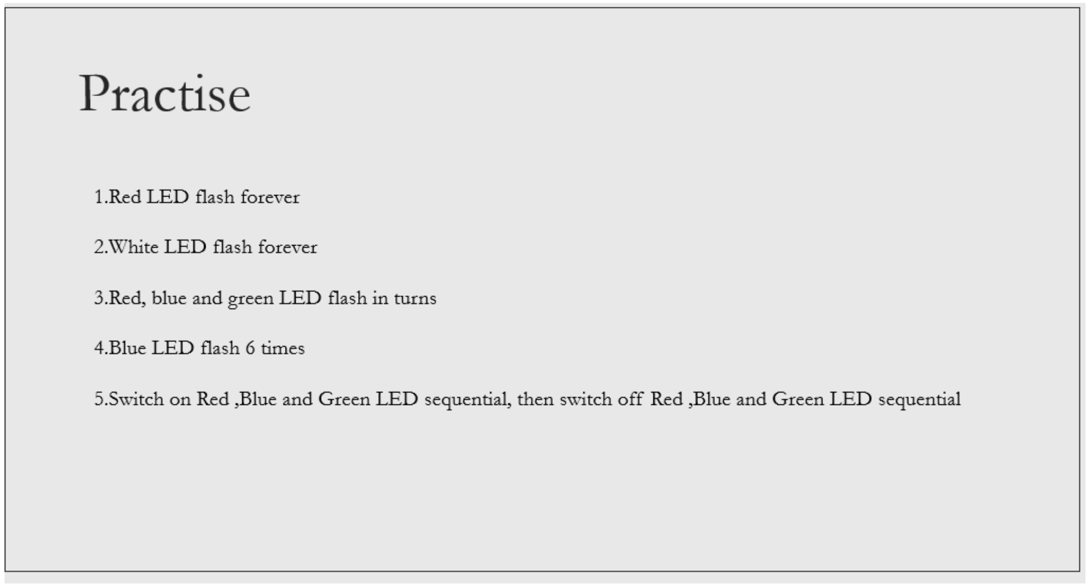
Switch on Green LED , Yellow LED in turns

- greenled=Greenled
- redled=Redled
- blueled=Blueled

  state=1 continue
- While(True)
- {
  - Switch state
  - {
    - Case 1:
    - greenled=On
    - Break

- Case 2:
  - greenled=Off
  - state=3
  - Break
- Case 3:
- redled=On
- greenled=On
- Break
- }
- Wait=0.6
- }

Could you switch yellow LED , red LED and blue LED in turns?



Practise

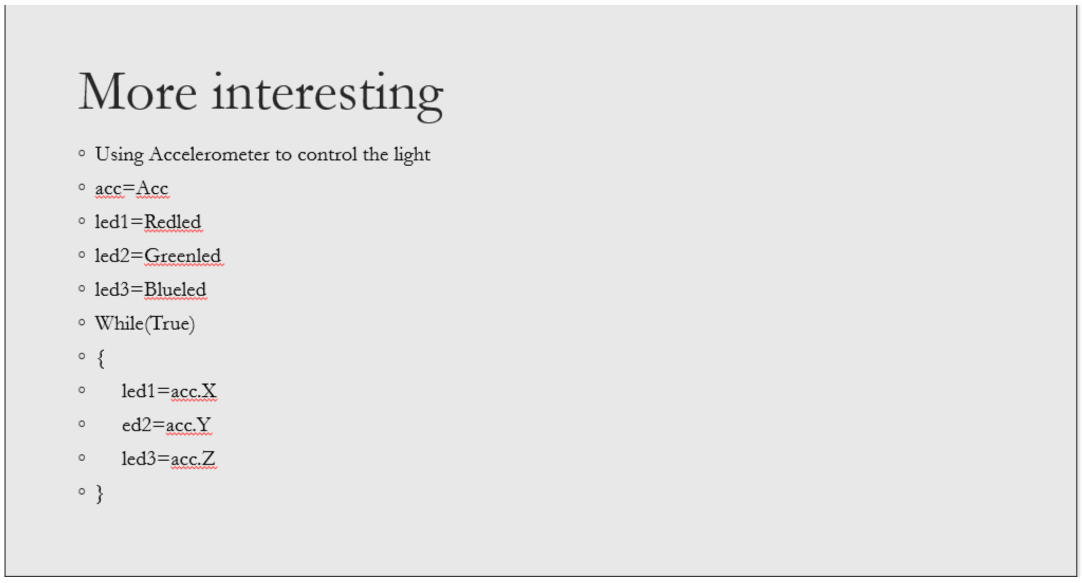
1.Red LED flash forever

2.White LED flash forever

3.Red, blue and green LED flash in turns

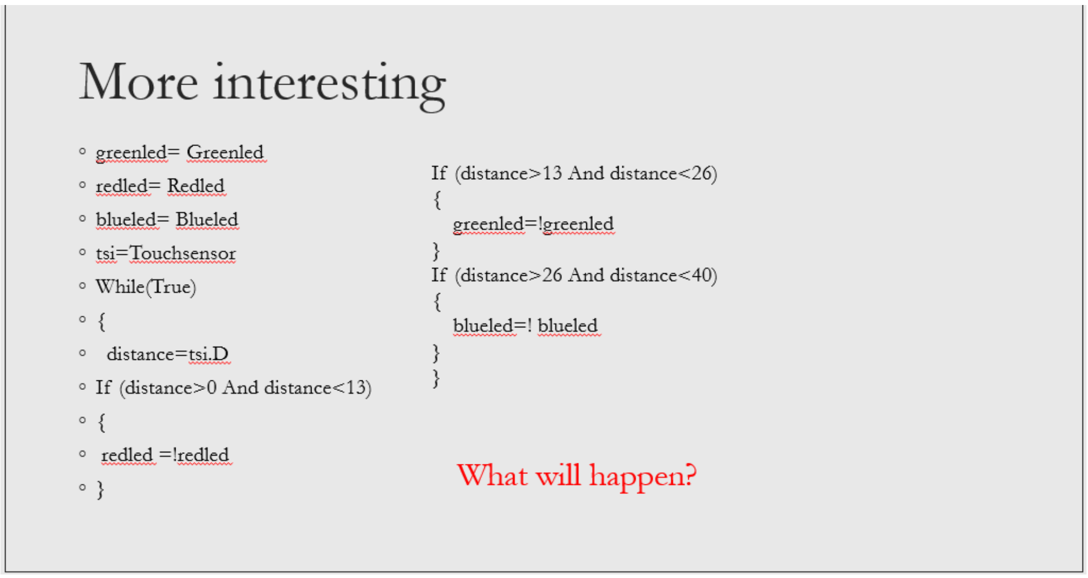
4.Blue LED flash 6 times

5.Switch on Red ,Blue and Green LED sequential, then switch off Red ,Blue and Green LED sequential



More interesting

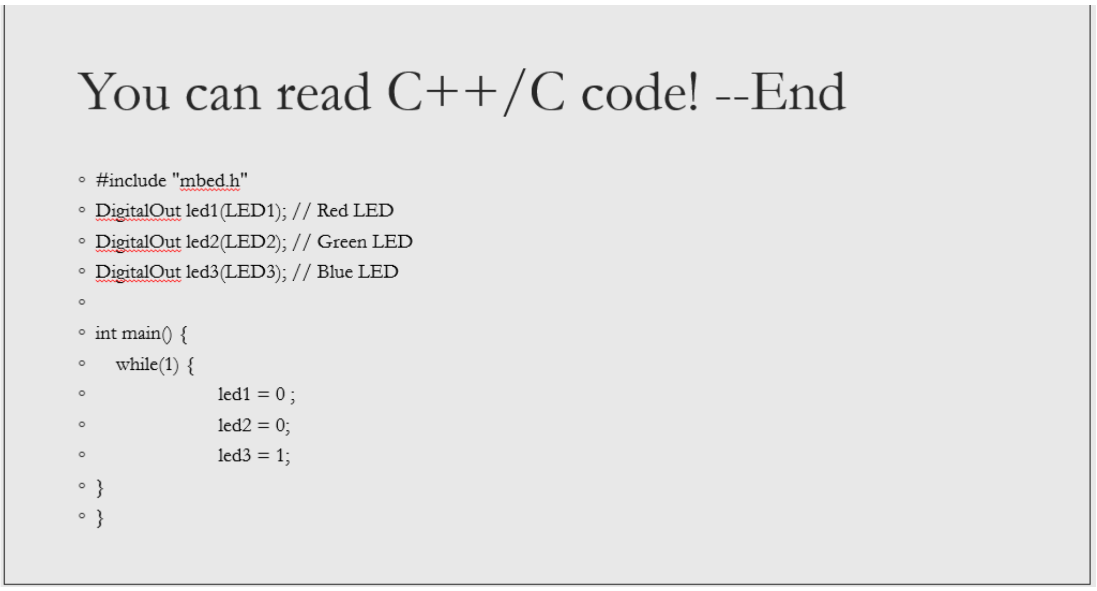
- Using Accelerometer to control the light
- acc=Acc
- led1=Redled
- led2=Greenled
- led3=Blueled
- While(True)
- {
- led1=acc.X
- ed2=acc.Y
- led3=acc.Z
- }



More interesting

```
◦ greenled= Greenled
◦ redled= Redled
◦ blueled= Blueled
◦ tsi=Touchsensor
◦ While(True)
◦ {
◦   distance=tsi.D
◦ If (distance>0 And distance<13)
◦ {
◦  redled =!redled
◦ }
If (distance>13 And distance<26)
{
   greenled=!greenled
}
If (distance>26 And distance<40)
{
   blueled=! blueled
}
}
```

What will happen?



You can read C++/C code! --End

```
◦ #include "mbed.h"
◦ DigitalOut led1(LED1); // Red LED
◦ DigitalOut led2(LED2); // Green LED
◦ DigitalOut led3(LED3); // Blue LED
◦
◦ int main() {
◦     while(1) {
◦                 led1 = 0 ;
◦                 led2 = 0;
◦                 led3 = 1;
◦ }
◦ }
```

A.3 Part of Measurement sheet

Case 1-Algorithm: How to switch on Blue LED or Red LED	
Spending time:	1 minutes
The level of difficulty:	Basic
The common errors:	Capital and lower case
Comment:	Very easy
Case 2-Algorithm: Mix two Red and Green LED to make purple colour	

Spending time:	1 minutes
The level of difficulty:	Basic
The common errors:	Using Wait
Comment:	Very easy
Case 3-Algorithm: Make green LED flash three times	
Spending time:	2 minutes
The level of difficulty:	Basic
The common errors:	Using Wait
Comment:	easy
Case 4-Repetition: Make green LED flash three times	
Spending time:	6 minutes
The level of difficulty:	Advance
The common errors:	Count
Comment:	Need help to explain counter
Case 5-Repetition: Make green LED flash forever	
Spending time:	4 minutes
The level of difficulty:	Basic
The common errors:	No
Comment:	Very cool
Case 6-Selection: Flash yellow LED and red LED in turns?	
Spending time:	10 minutes
The level of difficulty:	Advance
The common errors:	logic

Comment	Instead of using an If, use a list of algorithms instead.
Case 6-Selection: Flash yellow LED, red LED in turns?	
Spending time:	10 minutes
The level of difficulty:	Advance
The common errors:	Switch case syntax mistake
Comment	Although there is the harder to understand but children can use sample to finish it
Case 7-C++/C code: Read a simple C++ code to guess the function	
Spending time:	1 minutes
The level of difficulty:	Basic
The common errors:	Hard to accept the abstract hardware concept. For example, DigitalOut led2(LED2);
Comment	Although there is the harder to understand but children can use sample to finish it