

Kungliga Tekniska Högskolan
CSC
DD143X – Bachelor essay in computer science
Group examiner: Alexander Baltatzis

F# and Go compared to Java

With code implementations, benchmarking
tests and a survey study

Marcus Åberg
0707 32 85 58
Attundavägen 21a
168 58 Bromma
891028-0158
maraber@kth.se

Johan Lindeberg
0736 70 18 78
Ulvsvägen 8a
146 45 Stockholm
880112-0315
jlindebe@kth.se

Abstract

This report follows the Bachelor exam project followed through by two Computer Science in Engineering at the Royal Institute of Technology. The report presents the result from a study where a version of the breadth-first search algorithm was implemented and benchmarked in the two programming languages F# and Go. In addition to that the results from an online survey will be presented where other students studying Computer Science in Engineering at other Universities in Sweden answered questions regarding F# and Go.

F# and Go compared to Java

Bachelor exam report by Marcus Åberg and Johan Lindeberg

Table of Contents

F# and Go compared to Java.....	2
1 Introduction.....	4
2 Purpose.....	4
3 Background.....	4
3.1 The Java programming language.....	4
3.2 The Go programming language.....	5
3.2.1 General information.....	5
3.2.1 The Go syntax.....	6
3.3 The F# programming language.....	8
3.3.1 The F# syntax.....	8
3.4 Code readability.....	10
3.5 Benchmarking.....	10
4 Method.....	11
4.1 Code implementation.....	11
4.2 The benchmarking process.....	11
4.3 The survey.....	12
5 Results.....	12
5.1 Implemented code.....	12
5.2 Benchmark results.....	12
5.3 Survey results.....	14
6 Discussion.....	17
6.1 Survey.....	17
6.2 BFS.....	18
7 Conclusion.....	18
7.1 F#.....	18
7.2 Go.....	18
8 Bibliography.....	19
8.1 Source reference.....	19
Appendix 1.....	20
1.1 Implemented code in F#.....	20
1.2 Implemented code in Go.....	23

1.3 Implemented code in Java.....	26
1.4 The survey form	29
1.5 Master of Science in Engineering programs.....	43

1 Introduction

This report will handle a study of the two programming languages F# and Go. The reason these languages were chosen is their potential to become strong competitors of today's most popular way of programming.

F# was created by Microsoft as an improvement of the old programming language ML. By implementing the language into the Visual Studio environment it was given a world of possibilities, with resources such as a big library of predefined functions, a debugger and a code editor. As for Go which is a very new language (version 1 was recently released (March 28, 2012¹)) has not yet been given time to establish itself in the programming world. Though as for Google being one of the largest companies in the world and the creators of Go, one gets the notion that Go could become a very successful programming language if Google puts the effort into it.

After giving relevant background information about the two programming languages, a well-known algorithm that is called breadth-first search (BFS) will be implemented in the languages. In addition, the BFS will be implemented in Java, which will be used as background for the comparison that will be made between these languages.

2 Purpose

The main goals with this project are to find out about the differences and possible selling points of the F# and Go programming languages. This includes code readability, how hard it would be to learn these languages with previous knowledge of Java as well as some small-scale benchmarking. Java has been chosen since most of the Computer Science and Engineering (Civilingenjör i Datateknik) programs in Sweden include this language as a part of their course plan².

3 Background

This chapter informs the reader about the three different languages. Only a brief description will be given of Java, as the reader is presumed to have some previous knowledge of this language. In the Go and F# topics important syntax differences will be explained, and examples provided to give a better overview of these languages.

3.1 The Java programming language

Java is an object-oriented programming language which was released in 1995 by Sun Microsystems. It promised a "Write once, run anywhere"³ policy, with the intent to let the

¹ Gerrand, Andrew. Go version 1 released. 2012-03-28. <http://blog.golang.org/2012/03/go-version-1-is-released.html> (Accessed 2012-04-05)

² Appendix 1.3

³ Computer Weekly. Write once, run anywhere? May 2002. <http://www.computerweekly.com/feature/Write-once-run-anywhere> (Accessed 2012-04-01)

same code run over a variety of different platforms. This is achieved through the Java Virtual Machine (JVM), a program which runs between the Java program and the Operating System. JVM interprets the Java code to call the appropriate current Operating System commands.

Java quickly gained popularity, especially as web browsers incorporated the ability to run Java applets within web pages. Java has since then been one of the most popular programming languages, as seen in the below graph (Java in green).

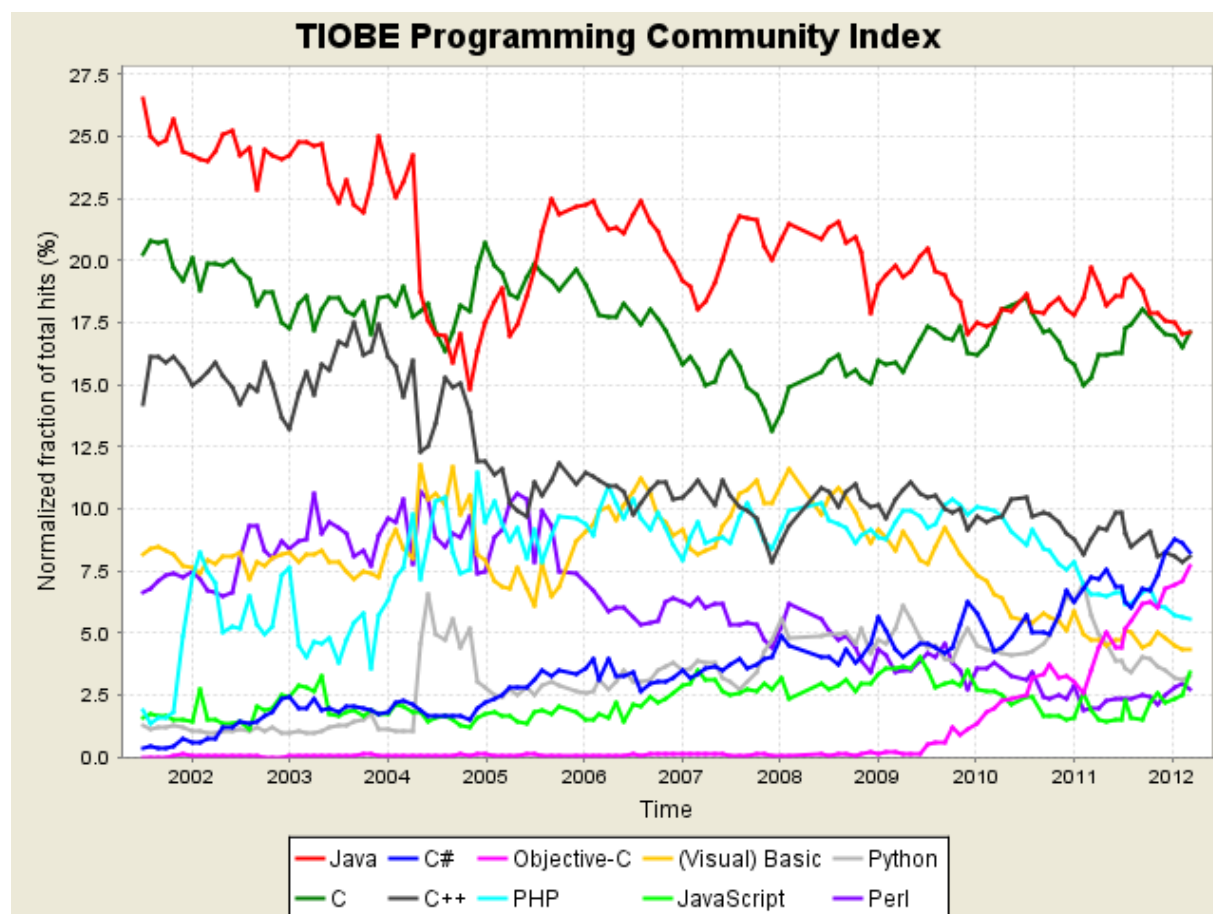


Figure 3.1⁴: TIOBE Programming community Index. Lists the popularity of different programming languages according to their definition⁵.

3.2 The Go programming language

3.2.1 General information

Go is an open source, general-purpose, compiled and concurrent programming language developed mainly by Google employees⁶. The development of the language started in

⁴ Paul Jensen. TIOBE Programming Community index for April 2012. 2012-04-08.

<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> (Accessed 2012-04-09)

⁵ Paul Jensen. TIOBE Programming Community Index Definition. 2012-04-08.

http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm (Accessed 2012-04-09)

⁶ Golang.org. The Go Programming Language Specification. 2012-03-17. <http://golang.org/ref/spec> (Accessed 2012-04-01)

September 2007 by Rob Pike, Robert Griesemer and Ken Thompson. Since it's an open source project a lot of people have contributed over the years - with code, documentations and ideas. In November 2009 the language was officially announced, and implementations made available for Linux, Mac OS X and Windows (although the Windows implementation had not yet been optimized). As of [March 28, 2012](#) Go version 1 (or Go 1 for short) was released, the first version of Go available in supported binary distributions across multiple platforms⁷. With this release a set of core libraries were defined to provide a stable foundation for creating reliable programs.

Go is object-oriented, as you can declare structs and make methods for these. Inheritance however is not supported, so there is no type hierarchy. Actually, there are no classes or subclasses at all⁸, and one could argue this is a big part of object-oriented programming.

Go is intended for program structure, not maximum performance. *“And finally, the emphasis in Go is on concurrent programming rather than parallel programming”* Russ Cox states⁹. By this Cox means that Go is focused on structuring your program in a way so that it can cope with having to do many things at once, but still let you write a simple, well-structured program.

3.2.1 The Go syntax

3.2.1.1 Types, variable declarations and interfaces

Types in Go are often inferred, not declared. Inside a Go function you can omit the type declaration when initializing a variable, and Go will automatically pull the type off of the initializer. This goes in line with the way interfaces are used in Go. Objects in Go satisfies an interface just by implementing the methods that the interface defines. Explicitly declaring that the object is intended to satisfy an interface is not needed.

A regular variable declaration creates a variable and binds an identifier to the variable, as in Java. If no value is given to the variable during declaration it is initialized to its zero value

```
var i int
var i int = 0
```

Both these lines of code initialize i as an int with the value zero.

After

```
type T struct { i int; f float64; next *T }
t := new(T)
```

The following holds:

⁷ Gerrand, Andrew. Go version 1 released. 2012-03-28. <http://blog.golang.org/2012/03/go-version-1-is-released.html> (Accessed 2012-04-05)

⁸ Pike, Rob and Cox, Russ. Google I/O 2010 – Go programming (Video). 2010-05-19. <http://www.google.com/events/io/2010/sessions/go-programming.html> (Accessed 2012-04-02)

⁹ Pike, Rob and Cox, Russ. Google I/O 2010 – Go programming (Video). 2010-05-19. <http://www.google.com/events/io/2010/sessions/go-programming.html> (Accessed 2012-04-02)

```
t.i == 0
t.f == 0.0
t.next == nil
```

As you can see, all the variables in *t* are initialized to their respective zero-values.

Go also introduces “Short variable declarations”, which may only appear inside functions. Unlike regular variable declarations, this type of declaration requires a value to be specified. The variable is initialized to the type of the assigned value.

```
i := 0
f := func() int { return 7 }
aString := "hello"
```

var i int = 0, var f int = 7, var aString string = "hello"

An interface type specifies a set of methods. Any type *T* which implements these methods are said to *implement the interface*. As stated earlier, no explicit declaration of this is needed.

```
type Lock interface {
    Lock()
    Unlock()
}
//The following two lines are method declarations
//which will be explained in 3.2.1.2
func (p T) Lock() { ... }
func (p T) Unlock() { ... }
```

With these few lines of code the type T will implement the interface Lock, since T have the methods Lock() and Unlock() defined.

3.2.1.2 Functions and methods

A function declaration binds an identifier (the function name) to a function. First comes the *func* keyword, followed by the name of the function and input variables within brackets. After this the return type is stated, followed by the function body within curly brackets. A function may omit the body, this means that the function has been implemented externally, such as an assembly routine.

```
func nameOfFunction(variable1 int, variable2 int) int {
    return variable1+variable2
}
```

```
func assemblyRoutine(x int) // body omitted, implemented externally
```

nameOfFunction takes two int variables, variable1 and variable 2 and returns an int. assemblyRoutine has no curly brackets, which is a sign that the function has been implemented externally

A method in Go is a function with a receiver. The receiver must be of type *T* or **T*, where *T* is a type name.


```
func (x *T) addOne() {
    x.value += 1;
}
```

*This shows how a method is created. x is the receiver, and has the type *T.*

If we now create a struct of type T with a value variable the addOne() function declared above can easily be called using the selector operation, as seen below.

```
type T struct {
    value int
}

func (x *T) addOne() {
    x.value += 1;
}
```

```
var z T //declares the variable z of type T. z.value is initialized to 0, as
no //value is given
z.addOne() //calls the method addOne, which increases z.value by one
```

3.3 The F# programming language

3.3.1 General information

F# is a hybridized programming language, crossbred from the best types in the programming world, object oriented and functional programming, implemented into the .NET environment. The language originates from the functional programming language ML¹⁰, which influenced several other programming languages, such as Standard ML, Haskell, C++, Caml and several others. As F# is an enhanced dialect of ML, it enables a user to copy ML written code straight into F#, and it will compile and run without any problems. Though that feature may only work one way, as many modern F# programmers use functions from the .NET library that is not implemented in ML. Also, the implementation of F# into Visual Studio's .NET framework allows F# to function freely with the other languages implemented into Visual Studio such as C/C++, C# and Visual Basic¹¹. The implementation also allows F# developers to access a very large library and set of tools provided when using Visual Studio, which is great as the library contains a huge amount of data structures and functions.

3.3.1 The F# syntax

3.3.1.1 White space based syntax

F# has a white space based syntax. It means that its syntax is based on new line, indentations and spaces. Compared to Java, where curly braces are required for the code within the body of each new function, code within an F# function that is limited to one row is

¹⁰ The SML/NJ Fellowship. Standard ML of New Jersey. 2004. <http://smlnj.sourceforge.net/> (Accessed 2012-03-16)

¹¹ Microsoft Research. F# at Microsoft Research. Date not available. <http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/> (Accessed 2012-03-16)

simply places on the right side of the equal sign. If the code within the body of the function is longer than one row, or for a more readable code a new line and an indentation is equal as how to define the body of a function. Another case is when the function has several commands of code to execute. Each command in the body has to be put on a new row or separated with a comma as the compiler interprets each line of code until it reaches a comma within a body as an execution command. See the examples following in *Code example 3.3.1.1.1* and *Code example 3.3.1.1.2*.

Code example 3.3.1.1.1

Allowed:

```
let executeThisAndThat ()=
    executeThis()
    executeThat()
```

Not allowed(as there is no comma to separate the commands:

```
let executeThisAndThat() =
    executeThis() executeThat()
```

Code example 3.3.1.1.2

A function with multiple bodies expressed with if clauses:

```
let checkIfTeenager x =
    if x >= 13
        then if x <= 19
            then true
```

3.3.1.2 Type inference

A few other things that is interesting to point out is the type inference¹² shown in *Code example 3.3.1.1.1* and how it affects the syntax. One does not have to declare of what type a variable is unless the compiler cannot decide of which type the variable derives from. This also means that the programmer does not have to declare a return type as the compiler recognizes a return clause and its return type. In *Code example 3.3.1.1.2* the type of input parameter “x” is not declared but the compiler assumes it is a variable of type integer as a boolean comparison is made between x and an integer.

3.3.1.3 Loops and boolean expressions

Another example shown in *Code example 3.3.1.3.1* is the omitted parenthesis wrapping the boolean expression. F# does not use parenthesis around boolean expressions and neither around the for-loops clauses.

Code example 3.3.1.3.1

for..to:

```
for i = 1 to 10
    do System.Console.Write(i)
```

for..downto:

```
for i = 10 downto 1
    do System.Console.Write(i)
```

The syntax of the for-loops clauses is different from other languages loops. Except for F#'s equality of the “for each”-loop, F# tells the user to define an interval rather than a boolean

¹² Microsoft Developer Network. Type Inference (F#). Date not available. <http://msdn.microsoft.com/en-us/library/dd233180.aspx> (Accessed 2012-03-18)

expression to calculate the amount of loops to execute¹³. This is shown in the two loop examples in *Code example 3.3.1.3.1*

3.3.1.4 Lambda expressions

Lambda expressions are powerful operations mainly used on lists and collections. They can be used as substitutes for functions, saving time since lambda expressions dissolves the need for declaring functions as they can be used anywhere fit. In *Code example 3.3.1.4.1* an example is shown of how the lambda expression can be used for a more efficient way of coding.

Code example 3.3.1.4.1

Without lambda expression:

```
let square x = x*x
let squares = List.map square [1..10]
```

Using lambda expression:

```
let squares = List.map (fun x -> x*x) [1..10]
```

3.4 Code readability

The expression “code readability” will be used often when discussing the results of the survey. When referring to the expression in this report, it will simply be a question of “is one able to read and interpret the basics of the code?”. To understand the intention of code segments such as function parameters and return values, variables and their types, and classes with their properties. These qualities can be hard to comprehend for someone who has not studied or used the language previously. The reader would then have to guess the intuitive intention of the code based on his or hers former programming experiences in an attempt to establish code readability.

3.5 Benchmarking

There are several ways of conducting a benchmark. In this project the benchmark process will focus on taking time. How the time taking process works is that a snapshot of the system clock, referred to as clock 1, is taken when the benchmarking begins. Then another snapshot is taken when the benchmarking ends, referred to as clock 2. The time taken is then the result of clock 2 - clock 1, delivered in nano seconds.

Taking time when programming is very common, and in most languages there is already a defined class that can access the system, making it very simple to measure time when executing code.

¹³ Microsoft Developer Network. Loops: for...in Expression. Date not available. <http://msdn.microsoft.com/en-us/library/dd233227.aspx> (Accessed 2012-03-18)

4 Method

4.1 Code implementation

The code writing process followed the same pattern in all of the languages. First off, a small algorithm was implemented to make some values for the nodes in our graph. This algorithm will be referred to as “Make list” in the future.

After this a way to represent the nodes was required. In Java and F# this took the form of a class, while in Go (which does not support classes) it was represented as a struct.

To create the graph on which the BFS later was performed on, a “Make graph” function was implemented. “Make graph” takes values computed in the “Make list” function to create a graph (a list of node elements) for the BFS to run on.

The BFS itself was implemented according to a common BFS pseudo code¹⁴. This implementation of BFS uses a queue structure for unvisited nodes, and as Go had no package routine for a Queue structure, an open source Go implementation of Queue was used¹⁵.

4.2 The benchmarking process

When the group had implemented the algorithms separately into the different languages a stopwatch benchmarking was executed. The stopwatch benchmarking process was conducted by using a number of implementations of the stopwatch class found in the languages libraries. Each stopwatch was programmed to time a specified algorithm in a series of executions. This process was performed by creating a loop for each algorithm, calling it a specified number of times. The result was then divided by the value used in the loops in order to establish an average execution time for each call to the algorithm. In this study the value 10.000 were used to define the number of times to loop.

The study was conducted on two separated computers whose technical details are found in Figure 4.1, whereas the results of the benchmarking process are found in *5.2 Benchmark results*.

Computer	CPU	CPU Bus speed	DRAM Frequency
Computer 1	Intel i7 950 @ 3,8GHz	200 MHz	6Gb DDR3 RAM @ 800Mhz
Computer 2	AMD Phenom II Quad core 940 @ 3,0GHz	200 Mhz	6Gb DDR2 RAM @ 333Mhz

Figure 4.1: the technical details of the computers benchmarked on.

¹⁴ Not available. Wikipedia. 2012-04-12. http://en.wikipedia.org/wiki/Breadth-first_search#Pseudocode (Accessed 2012-04-12)

¹⁵ Not available. Rosettacode. 2012-02-08. <http://rosettacode.org/wiki/Queue/Definition#Go> (Accessed 2012-04-12)

4.3 The survey

From the experience gained when implementing the BFS in F# and Go a number of questions regarding syntax readability was decided upon. These questions were put into a multiple choice survey and sent out to the Computer Science departments of all the major universities in Sweden conducting Master of Science in Engineering programs. For each question a number of answers were given to choose from, were only one answer was correct.

A survey taker has to specify which University he or she is currently studying at, what year they are registered in, if they have had any previous experience with functional programming and if they have had any previous experience with Go or F#. Except from this information, the survey is totally anonymous. It was constructed using a survey tool which is offered when using Google Documents online and is found in Appendix section *1.5 Master of Science in Engineering programs*.

5 Results

5.1 Implemented code

All code implemented in F# is found in the Appendix at section *1.1 Implemented code in F#*, while all code implemented in Go is found in the Appendix at section *1.2 Implemented code in Go* and all code implemented in Java is found in the Appendix at section *1.3 Implemented code in Java*

5.2 Benchmark results

Computer 1, time in ms	Make list	Create graph	Run BFS	Everything
F#	28,87	41,86	1,50	72,23
Go	0,05	0,179	0,753	0,982
Java	0,006	0,144	0,175	0,325

Table 5.2.1: Computer 1 test results

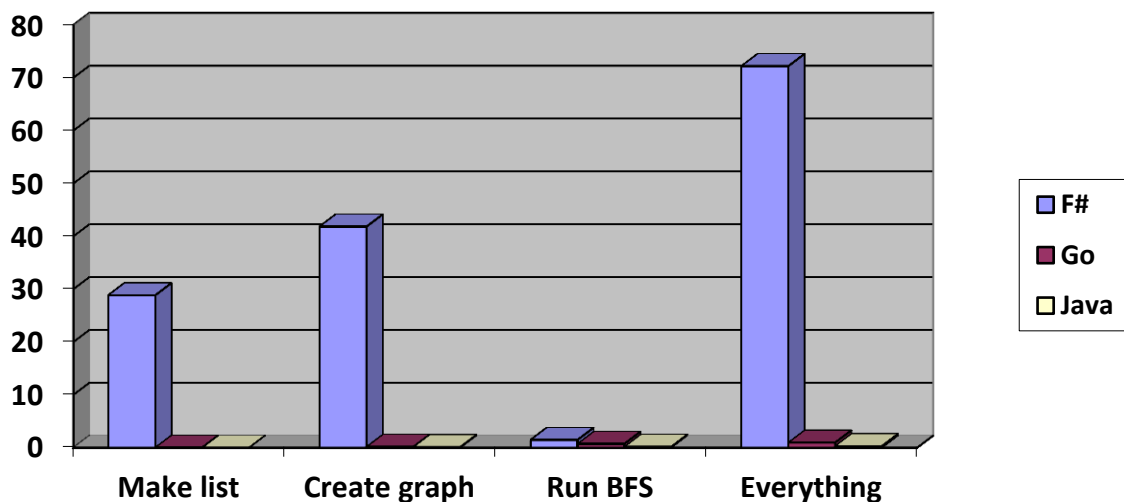


Chart 5.2.1: Computer 1 test results

Computer 2, time in ms	Make list	Create graph	Run BFS	Everything
F#	54,57918572	76,56771481	4,17422756	135,32212809
Go	0,038	0,266	0,852	1,156
Java	0,021	0,551	0,806	1,378

Table 5.2.2: Computer 2 test results

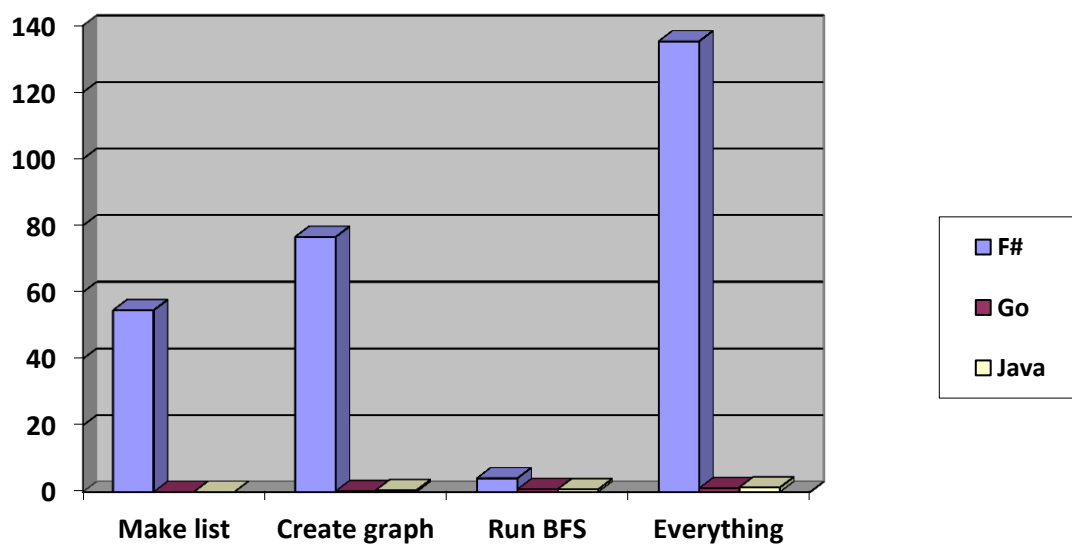


Chart 5.2.1: Computer 2 test results

5.3 Survey results

F#	Right answer
[1..10]	61
[x*x]-lista	56
F# map	38
Node, mutable	42
Change Nodes value to 1	9
Array.map(x, y)	49
Match, head-tail	30
List.map List.sum	36

Table 5.3.1: results from F# questions.

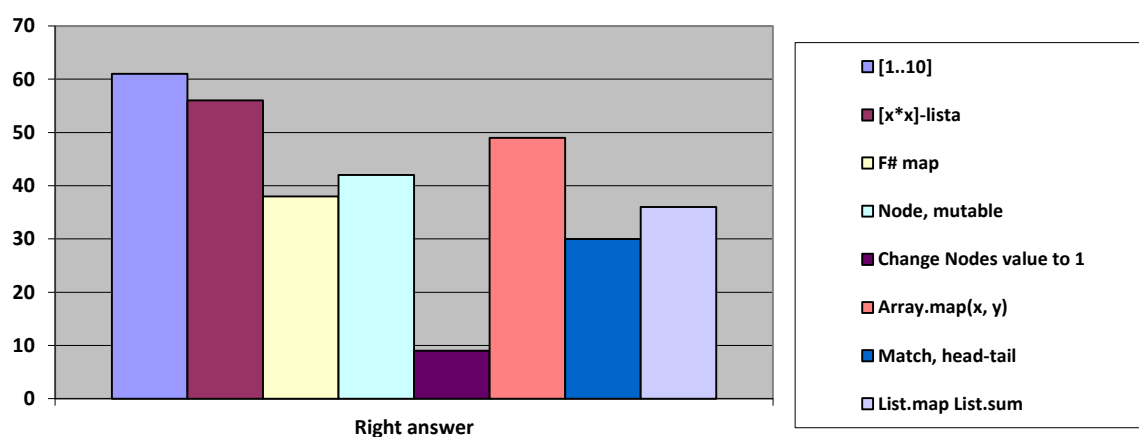


Chart 5.3.1: results from F# questions.

Go	Right answer
:= operator	43
Functions, parameters	47
Method	38
Structs	28
Array, functions	43
Go map function	51
:=, pointer, value	36

Table 5.3.2: results from Go questions.

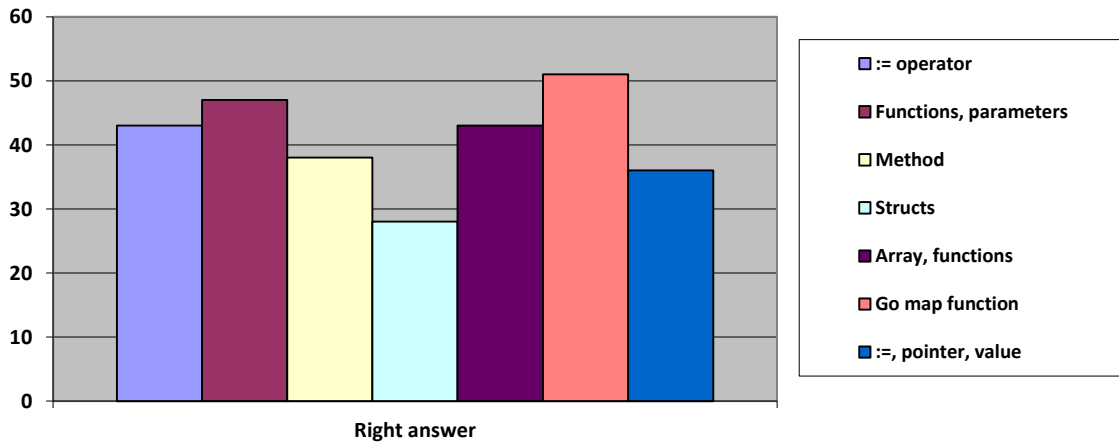
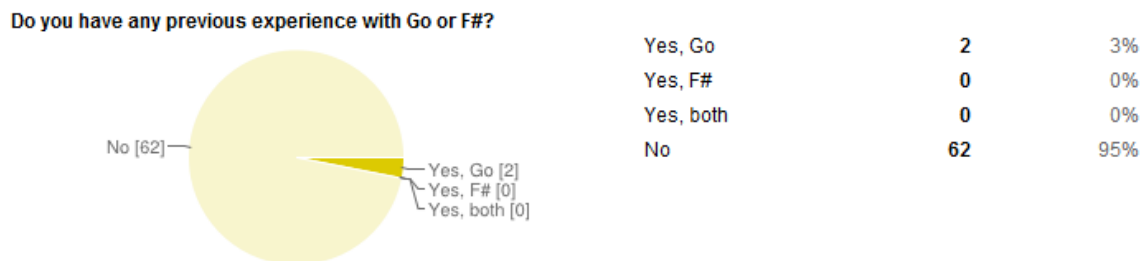
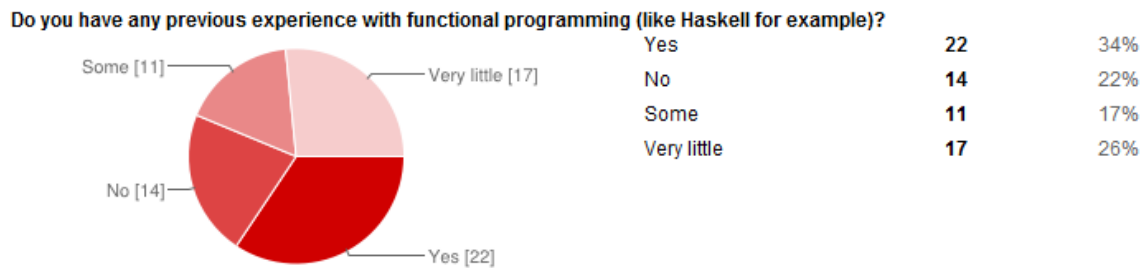
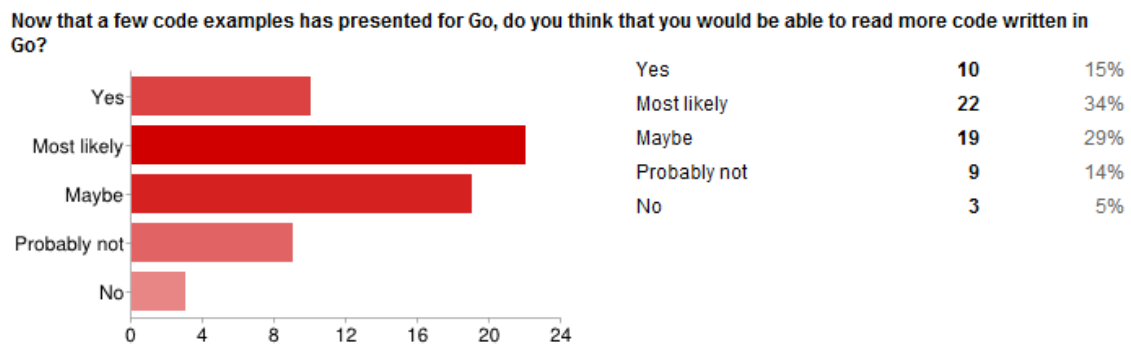
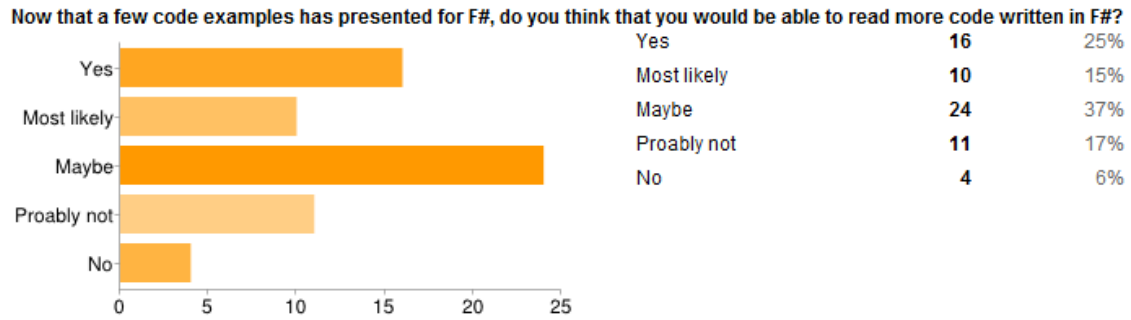


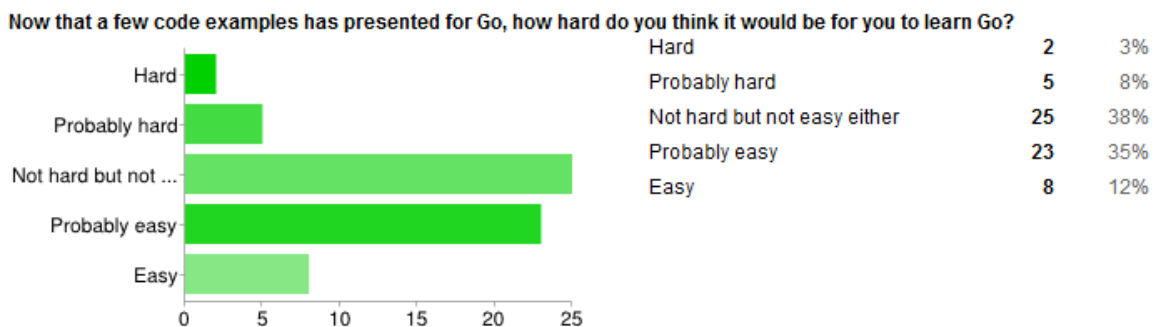
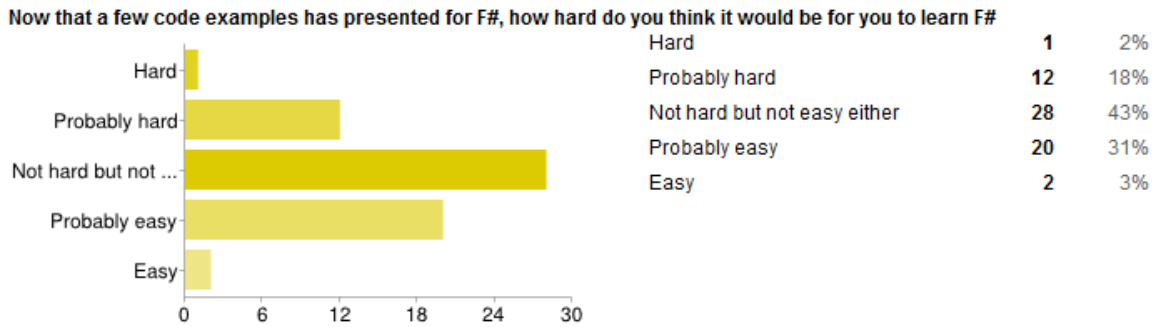
Chart 5.3.1: results from Go questions.



Picture 5.3.1: previous experiences



Picture 5.3.2: able to read F# and Go code in the future?



Picture 5.3.3: how hard would it be to learn F# and Go?

6 Discussion

6.1 Survey

The code examples for the questions in the survey was taken from various places such as from our BFS implementations and tutorials provided by Microsoft and Google for the different languages. As is, the questions may be of varying complexity - although an attempt was made trying to keep them at the same level. For this reason the results gathered from the survey can only be acknowledged as a general pointer to how easily F# and Go code is to read, and not an absolute truth. However, the survey participants were also asked if they think they would be able to read more code in F# and Go, and how hard they think it would be for them to learn the languages after taking the survey. This is a good starting point in the attempt to try and establish how complex it actually is to read and write code in these two languages.

No one in the survey had any previous knowledge of F#, yet, 25% of the participants stated that they think they would be able to read more code written in F#. On the other hand only 10% stated the same about Go where actually two answered that they had previous knowledge of Go.

The overall result of the “How hard would it be for you to learn F#” question was quite non-giving. As much as 43% stayed neutral to the question which suggests that most participants grasped at least the basics of the syntax as they did not answer “no” or “probably not”. Same

thing goes for Go where 39% stated that they were neutral to the question, although a much higher per cent thought it would probably be easy to learn Go rather than F#.

The only part in the Go-section of the survey where more participants stated the wrong answer rather than the right was regarding structs and inferred type declarations. This is a very distinct part of the Go language, and was an easy task for us to learn.

6.2 BFS

The short amount of time for this project did not let us fully learn and comprehend the way of programming in the new languages. Because of this, the implementations of the BFS in F# and Go are most likely not optimal, while the implementation in Java has a higher chance to be so. Especially F# that has a very different approach to programming than what Java students are used to and this is probably the main reason why the F# implementation ran slower than in Java and Go. Since Java has been around for over a decade and gone through a lot of work, the compiler probably optimizes even poorly written code quite well. Even though F# and Go has companies such as Microsoft and Google working on them, they are still relatively new languages and therefore have not yet had the chance to evolve in the same matter as Java.

7 Conclusion

As both F# and Go are backed up by two of the leading software companies in today's industry, they certainly have a possibility to become competitors of today's big programming languages.

Most participants gave the right answer for the code examples given in the survey and both languages seems quite understandable for students currently studying Computer Science and Engineering.

7.1 F#

As we state in the discussion, the poor performance result of the F# benchmark is probably due to our inexperience with functional programming. However, this shows that F# is quite hard to learn with mostly a background of Java programming. Although it might be hard to learn, reading the code seems quite easy as stated earlier. Only 21% of the participants stated that they would probably not, or would not, be able to read more code written in F#. Most of the participants in the survey had no opinion about how hard the language would be to learn.

7.2 Go

Go as a language is more closely related to our previous programming experience, and therefore writing the BFS was a much easier task in Go compared to F#. This also shows in the results of the benchmarking, as Go and Java performed quite equally. The results of the survey tends towards Go being "*Probably easy*" to learn and overall, the participants stated mostly the right answers to the questions given. Our opinion falls the same way, and Go code is quite enjoyable to read.

8 Bibliography

8.1 Source reference

1. Gerrand, Andrew. Go version 1 released. 2012-03-28.
<http://blog.golang.org/2012/03/go-version-1-is-released.html> (Accessed 2012-04-05)
2. Appendix REGERGREUGERUGWEUFUUQWE
3. Computer Weekly. Write once, run anywhere? May 2002.
<http://www.computerweekly.com/feature/Write-once-run-anywhere> (Accessed 2012-04-01)
4. Paul Jensen. TIOBE Programming Community index for April 2012. 2012-04-08.
<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> (Accessed 2012-04-09)
5. Tiobe Software. TIOBE Programming Community Index Definition. Date not available.
http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm (Accessed 2012-04-09)
6. Golang.org. The Go Programming Language Specification. 2012-03-17.
<http://golang.org/ref/spec> (Accessed 2012-04-01)
7. Gerrand, Andrew. Go version 1 released. 2012-03-28.
<http://blog.golang.org/2012/03/go-version-1-is-released.html> (Accessed 2012-04-05)
8. Pike, Rob and Cox, Russ. Google I/O 2010 – Go programming (Video). 2010-05-19.
<http://www.google.com/events/io/2010/sessions/go-programming.html> (Accessed 2012-04-02)
9. Pike, Rob and Cox, Russ. Google I/O 2010 – Go programming (Video). 2010-05-19.
<http://www.google.com/events/io/2010/sessions/go-programming.html> (Accessed 2012-04-02)
10. The SML/NJ Fellowship. Standard ML of New Jersey. 2004.
<http://smlnj.sourceforge.net/> (Accessed 2012-03-16)
11. Microsoft Research. F# at Microsoft Research. Date not available.
<http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/> (Accessed 2012-03-16)
12. Microsoft Developer Network. Type Inference (F#). Date not available.
<http://msdn.microsoft.com/en-us/library/dd233180.aspx> (Accessed 2012-03-18)
13. Microsoft Developer Network. Loops: for...in Expression. Date not available.
<http://msdn.microsoft.com/en-us/library/dd233227.aspx> (Accessed 2012-03-18)
14. Not available. Wikipedia. 2012-04-12. http://en.wikipedia.org/wiki/Breadth-first_search#Pseudocode (Accessed 2012-04-12)
15. Not available. Rosettacode. 2012-02-08.
<http://rosettacode.org/wiki/Queue/Definition#Go> (Accessed 2012-04-12)

Appendix 1

1.1 Implemented code in F#

```

module VerticeClass

open System;

type Vertice (value: int, adjVerticePointers:int array) =
    //Where "adjVerticePointers" has the index value of the number "value" was made
    from.
    //In other words, "value", has the value of the number located at index i when
    creating the Vertice objects from list "List"
    new (value) = Vertice(value)
    new (value, adjVerticePointers) = Vertice(value, adjVerticePointers)

    member this.value = value
    member this.adjVerticePointers:int array = adjVerticePointers

    override this.ToString() =
        if adjVerticePointers.Length = 0 then (string)value
        else (string)value + " " + (string)adjVerticePointers.[0] + " " +
            (string)adjVerticePointers.[1]

module NumberGenerator

open System;

//Creates numbers 1,2,4,7,11,16... n times
let rec helpToPopulateArray n count number (myArray:int array) =
    if count < n then helpToPopulateArray n (count+1) (number+count) (Array.append myArray
[|number+count|])
    else myArray

let populateArray n = helpToPopulateArray n 1 1 [|1|]

```

```

module Main

open System;
open System.Diagnostics;
open System.Collections.Generic;
open VerticeClass;
open NumberGenerator;

let number = 10000
let listOfNumbers:int array = populateArray number
let endVertice = 48595012 //Vertice nr 9858
let VerticeQueue = new Queue<(Vertice)>()

//Help method for MakeVerticeList
let rec ConnectEndVertices (listOfNumbers:int array) indexCount (verticeList: Vertice
array) =
    if (indexCount < listOfNumbers.Length)
    then ConnectEndVertices (listOfNumbers:int array) (indexCount + 1) (Array.append
verticeList [|Vertice(listOfNumbers.[indexCount], [||])|])
    else verticeList

//indexCount remembers on which index in listOfNumbers to get the value
//treeLevel tells us at which level we are in the tree, verticeList is the result of the
function
//Call MakeVerticeList with 0, 1, 1, []
let rec MakeVerticeList indexCount currentTreeLevel numbersLeftBeforeLevelChange
(verticeList:Vertice array) =
    if indexCount < (listOfNumbers.Length - currentTreeLevel - 1)
    then if numbersLeftBeforeLevelChange = 0
        then MakeVerticeList indexCount (currentTreeLevel + 1) (currentTreeLevel + 1)
(verticeList:Vertice array)
        //Still have numbers to cover on this level
        else MakeVerticeList (indexCount + 1) currentTreeLevel
(numbersLeftBeforeLevelChange - 1)
(Array.append verticeList [|Vertice(listOfNumbers.[indexCount],
[|listOfNumbers.[(indexCount + currentTreeLevel)]; listOfNumbers.[(indexCount +
currentTreeLevel + 1)]|])|])
    else ConnectEndVertices (listOfNumbers:int array) indexCount (verticeList: Vertice
array)

let verticeList = MakeVerticeList 0 1 1 [||]

//Queues all vertices in the interval startVertice to startVertice+treeLevel
let helpToQueueVertices indexFrom treeLevel =
    for i = indexFrom to treeLevel do
        VerticeQueue.Enqueue(Array.get verticeList i)

let helpToQueueVertices2 indexFrom indexTo =
    for i = indexFrom to indexTo do
        VerticeQueue.Enqueue(Array.get verticeList i)

```

```

helpToQueueVertices 0 0 //Enqueue the first vertice before starting the BFS
let rec doTheBFS indexFrom indexTo treeLevel verticesLeftInGraph =
    if verticeList.Length > indexTo
        then while VerticeQueue.Count > 0 do
            if VerticeQueue.Dequeue().value = endVertice then
                System.Console.WriteLine("I found your vertice!")

                if (Array.IndexOf(listOfNumbers, endVertice)) <= indexTo
                    then doTheBFS indexFrom verticeList.Length treeLevel
verticesLeftInGraph
                else helpToQueueVertices (indexFrom + treeLevel) ((indexFrom +
treeLevel + treeLevel))
                    doTheBFS (indexFrom + treeLevel) (indexFrom + treeLevel +
treeLevel) (treeLevel + 1) (number - treeLevel)

let MakeListStopwatch = new Stopwatch()
let MakeGraphStopwatch = new Stopwatch()
let DoBFSStopwatch = new Stopwatch()

MakeListStopwatch.Start()
let doListOfNumbersATHousandTimes =
    for i = 1 to 10000 do
        populateArray number
MakeListStopwatch.Stop()

MakeGraphStopwatch.Start()
let doVerticeListATHousandTimes =
    for i = 1 to 10000 do
        MakeVerticeList 0 1 1 [||]
MakeGraphStopwatch.Stop()

DoBFSStopwatch.Start()
let doBFSATHousandTimes =
    for i = 1 to 10000 do
        doTheBFS 0 0 1 number
DoBFSStopwatch.Stop()

System.Console.Write("The time of doListOfNumbersATHousandTimes took: ")
System.Console.WriteLine((string)(MakeListStopwatch.Elapsed) + " ")
System.Console.Write("The time of doVerticeListATHousandTimes took: ")
System.Console.WriteLine((string)MakeGraphStopwatch.Elapsed + " ")
System.Console.Write("The time of doBFSHunderedATHousandTimes took: ")
System.Console.WriteLine((string)DoBFSStopwatch.Elapsed + " ")

```

1.2 Implemented code in Go

```

package main

import (
    "fmt"
    "time"
)

type Node struct {
    value int64
    adjNodes [2]int
    visited bool
}

type Queue struct {
    b []Node
    head, tail int
}

//how many times to run the test
const AmountOfRuns = 100

func main() {
    //How many times the test is run

    //time variables
    var everythingTime time.Duration
    var listTime time.Duration
    var graphTime time.Duration
    var bfsTime time.Duration

    //test-loop
    for i:=0; i<AmountOfRuns; i++ {

        e1 := time.Now()
        //create some values for the graph
        l1 := time.Now()
        list := makeList(10000)
        l2 := time.Now()
        //build the graph with the values from list
        g1 := time.Now()
        graph := makeGraph(list)
        g2 := time.Now()
        b1 := time.Now()
        q := new(Queue)
        bfs(graph,q,49995001)
        b2 := time.Now()
        e2 := time.Now()

        everythingTime += e2.Sub(e1)
        listTime += l2.Sub(l1)
        graphTime += g2.Sub(g1)
        bfsTime += b2.Sub(b1)
    }

    //debug
    /*
    list := makeList(10000)

```



```

fmt.Println(list)

graph := makeGraph(list)
for i:=0;i < len(graph);i++ {
    fmt.Print(i+1)
    fmt.Print(": ")
    fmt.Println(graph[i])
}
*/

fmt.Println("Everything took: ", everythingTime/AmountOfRuns)
fmt.Println("Making list took: ", listTime/AmountOfRuns)
fmt.Println("Making graph took: ", graphTime/AmountOfRuns)
fmt.Println("Doing bfs took: ", bfsTime/AmountOfRuns)

}

func bfs (graph []Node, q *Queue, searchValue int64) bool {
    currentNode := graph[0]
    var notEmpty bool = true
    q.Push(currentNode)
    for ;notEmpty == true; {
        currentNode, notEmpty = q.Pop()
        if searchValue == currentNode.value {
            return true
        }
        for j:=0;j<len(currentNode.adjNodes);j++ {
            if !graph[currentNode.adjNodes[j]].visited {
                graph[currentNode.adjNodes[j]].visited = true
                q.Push(graph[currentNode.adjNodes[j]-1])
            }
        }
    }
    return false
}

//Returns an array of Nodes with the values from list
func makeGraph (list []int64) []Node {
    g := make([]Node, len(list)*2)

    g[0].value = 1
    g[0].adjNodes[0] = 2
    g[0].adjNodes[1] = 3
    j := 3
    for i:=1;i<len(list);i++ {
        g[i].value = list[i]
        j++
        g[i].adjNodes[0] = j
        j++
        g[i].adjNodes[1] = j;
    }
    return g
}

//Returns a list of values of size x
func makeList(x int64) []int64 {
    var number int64 = 1
    var count int64 = 1
    list := make([]int64, x)
    list[0] = 1

```

```

    for ; count != x ; {
        list[count] = (number + count)
        number += count
        count++
    }
    return list
}

// Queue from http://rosettacode.org/wiki/Queue/Definition#Go
// the zero object is a valid queue ready to be used.
// int queue
// items are pushed at tail, popped at head.
// tail = -1 means queue is full
func (q *Queue) Push(x Node) {
    switch {
    // buffer full. reallocate.
    case q.tail < 0:
        next := len(q.b)
        bigger := make([]Node, 2*next)
        copy(bigger[copy(bigger, q.b[q.head:]):], q.b[:q.head])
        bigger[next] = x
        q.b, q.head, q.tail = bigger, 0, next+1
    // zero object. make initial allocation.
    case len(q.b) == 0:
        q.b, q.head, q.tail = make([]Node, 10), 0, 1
        q.b[0] = x
    // normal case
    default:
        q.b[q.tail] = x
        q.tail++
        if q.tail == len(q.b) {
            q.tail = 0
        }
        if q.tail == q.head {
            q.tail = -1
        }
    }
}

func (q *Queue) Pop() (Node, bool) {
    if q.head == q.tail {
        z := Node{}
        return z, false
    }
    r := q.b[q.head]
    if q.tail == -1 {
        q.tail = q.head
    }
    q.head++
    if q.head == len(q.b) {
        q.head = 0
    }
    return r, true
}

func (q *Queue) Empty() bool {
    return q.head == q.tail
}

```

1.3 Implemented code in Java

Node.java

```
public class Node {
    long value;
    int[] neighbours;
    boolean visited;

    //A node in the graph. Initializes two edges to neighbouring nodes
    //a value for the node and sets visited to false
    public Node(long value, int left, int right) {
        this.value = value;
        this.neighbours = new int[2];
        this.neighbours[0] = left;
        this.neighbours[1] = right;
        this.visited = false;
    }
}
```

testBfs.java

```
import java.util.LinkedList;
import java.util.Queue;

public class testBfs {

    //number of runs for the benchmark
    public static final int NUMBER_OF_RUNS = 100;
    //size of the graph
    public static final int SIZE = 10000;

    public static void main(String[] args) {

        long listT = 0;
        long graphT = 0;
        long bfsT = 0;

        //the benchmark loop, times the different parts of the program
        //with System.nanoTime()
        for (int i=0; i<NUMBER_OF_RUNS; i++) {

            long l0 = System.nanoTime();
            int[] list = makeList(SIZE);
            long l1 = System.nanoTime();

            long g0 = System.nanoTime();
            Node[] graph = makeGraph(list);
            long g1 = System.nanoTime();

            long b0 = System.nanoTime();
            boolean search = bfs(graph, list[SIZE - 1]);
            long b1 = System.nanoTime();

            listT += (l1-l0);
            graphT += (g1-g0);
        }
    }
}
```

```

        bfsT += (b1-b0);
    }

    System.out.println("Make list: " + listT/NUMBER_OF_RUNS );
    System.out.println("Make graph: " + graphT/NUMBER_OF_RUNS );
    System.out.println("Do bfs: " + bfsT/NUMBER_OF_RUNS );
    System.out.println("Everything: " +
(listT+graphT+bfsT)/NUMBER_OF_RUNS);

}

//The breadth first search algorithm
public static boolean bfs(Node[] graph, int searchValue) {

    LinkedList<Node> q = new LinkedList();

    Node currentNode = graph[0];

    q.add(currentNode);

    boolean notEmpty = true;
    while(notEmpty==true) {

        currentNode = q.remove();
        if (searchValue == currentNode.value) {
            //System.out.println(currentNode.value);
            return true;
        }
        if (graph[currentNode.neighbours[0] - 1].visited == false) {
            graph[currentNode.neighbours[0] - 1].visited = true;
            q.add(graph[currentNode.neighbours[0] - 1]);
        }
        if (graph[currentNode.neighbours[1] - 1].visited == false) {
            graph[currentNode.neighbours[1] - 1].visited = true;
            q.add(graph[currentNode.neighbours[1] - 1]);
        }
    }

    return false;
}

//Creates a list with values for the graph
public static int[] makeList(int x) {
    int number = 1;
    int count = 1;
    int[] list = new int[x];

    for (;count!=x;) {
        list[count] = (number+count);
        number += count;
        count++;
    }
    return list;
}

//Initializes the graph
public static Node[] makeGraph(int[] list) {
    Node[] graph = new Node[list.length*2];

```

```
Node base = new Node(1,2,3);
graph[0] = base;

int j = 3;
for (int i=1;i<list.length; i++) {
    graph[i] = new Node(list[i], j, j+1);
    j+=2;
}
for (int i=list.length; i<list.length*2; i++) {
    graph[i] = new Node(0,0,0);
}
return graph;
}
}
```

1.4 The survey form

Survey F# and Go

At some point you will be given the answer to a previous question, please do not go back in the form to change an answer.

At which university/college are you studying at?

- Blekinge Tekniska Högskola
- Chalmers Tekniska Högskola
- Karlstad Universitet
- Kungliga Tekniska Högskolan
- Linköpings Tekniska Universitet
- Luleå Tekniska Universitet
- Lunds Tekniska Universitet
- Mittuniversitetet
- Umeå Högskola
- Uppsala Universitet

In which year are you currently registered in?

- Year 1
- Year 2
- Year 3
- Year 4
- Year 5

Do you have any previous experience with functional programming (like Haskell for example)?

- Yes
- No
- Some
- Very little

Do you have any previous experience with Go or F#?

- Yes, Go
- Yes, F#
- Yes, both
- No

Survey F# and Go

F# code readability

Here follows a couple of F# example code. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question.

Consider the code below. What do you think it does?

[1..10]

- Creates a list of some sort containing the elements [1, empty, empty, ..., 10]
- Creates a list of some sort containing the elements [1, 2, 3, ..., 10]
- Creates a list of some sort containing the elements [1, 10]
- Not sure

Consider the code below. What do you think it does?

[for x in 0..10 -> x*x]

- Creates a list in which each index contains a function
- Creates a list in which each index contains the return value of the function $x*x$
- Creates a list that contains the values [0*0, empty, empty, ..., 10*10]
- Creates a list that contains the values [0*0, 10*10]
- Not sure

Consider the code below. What do you think it does?

Array.map(x y)

- Creates a map, with x and y coordinates, of the size given by x and y
- Creates an array containing the elements of x plus the elements of y
- Creates an array containing the elements according to the pattern [x[0], y[0], x[1], y[1], ..., x[n], y[n]]
- Takes a function x and applies it on every element in y
- Not sure

Survey F# and Go

F# code readability

Here follows an example of an F# class. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question.

```
type Node(value, visited) =  
    let mutable internalValue= value  
    new (value) member this.value =  
        with get() = internalValue  
        and set(value) = internalValue <- value
```

Consider the code in the header of this page. Which row is the constructor of the class?

- type Node(value, visited) =
- let mutable internalValue= value
- new (value)
- with get() = internalValue
- and set(value) = internalValue <- value
- Not sure

Consider the code in the header of this page. One can create a new object of the Node class by writing for example "let rootNode = new Node(0, false)", but what would someone have to write in order to change the Nodes value to 1?

- let rootNode = Node(1, false)
- rootNode.value = 1
- rootNode.internalValue = 1
- rootNode.value <- 1
- rootNode.internalValue <- 1
- Not sure

Survey F# and Go

F# code readability

Here follows an example of F# code. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question.

```
let testArray = [|1;2;3;4;5|]
let square x = x*x
Array.map(square testArray)
```

Consider the code in the header of the page. What do you think the code "Array.map(x y)" does?

- Creates a map, with x and y coordinates, of the size given by x and y
- Creates an array containing the elements of x plus the elements of y
- Creates an array containing the elements according to the pattern $[x[0],y[0],x[1],y[1],\dots,x[n],y[n]]$
- Takes a function x and applies it on every element in y
- Not sure

Survey F# and Go

F# code readability

Here follows an example of F# code. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question.

```
let rec SomeFunction x =  
    match x with  
    | [] -> 0  
    | y::ys -> y + SomeFunction ys
```

Consider the code in the header of the page. What do you think the function does if x contains a value?

- It takes a word and returns the first letter
- It takes an integer and creates a list of size x
- It removes all zeroes from the list
- Takes a list x and returns the sum of the elements in x
- Not sure

Survey F# and Go

F# code readability

Here follows an example of F# code. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question.

```
let rec SomeFunction x =  
  match x with  
  | [] -> 0  
  | y::ys -> y + SomeFunction ys
```

Consider the code in the header of the page. What do you think the function does if x contains a value?

- It takes a word and returns the first letter
- It takes an integer and creates a list of size x
- It removes all zeroes from the list
- Takes a list x and returns the sum of the elements in x
- Not sure

Survey F# and Go

F# code readability

Here follows an example of F# code. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question.

```
let test n = [1..n]
|> List.map Square
|> List.sum
```

Consider the code in the header of the page. What do you think the function does?

- It sums the result and then squares the sum
- It squares each element and then sums the squares
- It returns an integer containing the value of the sum function and a list containing the squares of each element
- It says that only the functions “square” and “sum” are applicable on the list
- Not sure

Survey F# and Go

Go code readability

Here follows a couple of examples of Go code. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question.

Consider the code below. Which answer is correct?

```
list := makeList(100)
```

- list is initialized to the same type as the returned value from makeList(100), and holds the returned value.
- list has been initialized earlier, and holds the returned value of makeList(100)
- Not sure

Consider the code below. Which answer is correct?

```
func someFunction(graph []Node) bool
```

- someFunction takes some variable of type []Node and returns nothing
- someFunction takes a specific variable named graph and returns a bool
- someFunction takes some variable of type []Node and returns a bool
- someFunction takes a variable of type bool and returns a variable graph of type []Node
- Not sure

Consider the code below. On what struct / type is the method performed, what variables does it take and what is the return type?

```
func (q *Queue) Push(x Node) bool {code omitted}
```

- Performed on struct / type: bool, Input variable: x, Return type *Queue
- Performed on struct / type: *Queue, Input variable: bool, Return type: Node
- Performed on struct / type: Node, Input variable: x, Return type: bool
- Performed on struct / type: Node, Input variable: q, Return type: bool
- Performed on struct / type: *Queue, Input variable: x, Return type: bool
- Performed on struct / type: bool, Input variable: q, Return type: Node

Consider the code below. What values does r hold after execution?

```
type Vertex struct { X, Y int } r = Vertex{X: 1}
```

- {1,0}

- {0,1}
- Uncertain

Survey F# and Go

Go code readability

Here follows an example of Go code. Just check the option that you think is the right one, and if you do not know just check the bottom option of the question. `fmt.Println()` is a standard package function to print text.

```
x:= []int{0,1,2,3,4,5,6,7,8,9}
for i:=0; i<10; i++ {
    x[i] = x[i]*x[i]
}
fmt.Println(x[0:5])
```

Consider the code in the header of the page. What do you think is displayed after execution?

- {0,1,4,9,16,25,36,49,64,81}
- {0,1,4,9,16,25}
- {5,6,7,8,9}
- {0:5}
- Not sure

Survey F# and Go

Go code readability

Here follows an example of Go code. What do you think is printed out on screen after execution?

```
m = make(map[string]Vertex)
m["Bell Labs"] = Vertex{
    40.68433, 74.39967
}
fmt.Println(m["Bell Labs"])
```

Go map. What is printed out after execution?

- {40.68433 74.39967}
- Bell labs {40.68433 74.39967}
- Bell labs
- {40.68433 74.39967} Bell labs
- Not sure

Survey F# and Go

Go code readability

Here follows an example of Go code. What do you think is printed out on screen after execution?

```
p := Vertex{1, 2}
q := &p
q.X = 1e9
fmt.Println(p)
```

What is printed out after execution?

- {0 2}
- {1000000000 0}
- {1000000000 2}
- {1,2}
- Not sure

Survey F# and Go

What did you think about F# and Go?

This is the last page and we would very much like to know what you thought about the two programming languages based on the example code that you have seen

Now that a few code examples has presented for F#, do you think that you would be able to read more code written in F#?

- Yes
- Most likely
- Maybe
- Probably not
- No

Now that a few code examples has presented for Go, do you think that you would be able to read more code written in Go?


- Yes
- Most likely
- Maybe
- Probably not
- No

Now that a few code examples has presented for F#, how hard do you think it would be for you to learn F#

- Hard
- Probably hard
- Not hard but not easy either
- Probably easy
- Easy

Now that a few code examples has presented for Go, how hard do you think it would be for you to learn Go?

- Hard
- Probably hard
- Not hard but not easy either
- Probably easy

-  Easy

1.5 Master of Science in Engineering programs

College/University	Studies Java	Program if other than Master of Science in Engineering
Blekinge Tekniska Högskola	No	
Chalmers Tekniska Högskola	Yes	
Karlstad Universitet	Yes	
Kungliga Tekniska Högskolan	Yes	
Linköpings Tekniska Universitet	Yes	
Luleå Tekniska Universitet	Yes	
Lunds Tekniska Högskola	Yes	
Mittuniversitetet	Yes	
Umeå Högskola	Yes	Civilingenjörsprogrammet i Teknisk datavetenskap
Uppsala Universitet	Yes	Civilingenjörsprogrammet i Informationsteknologi