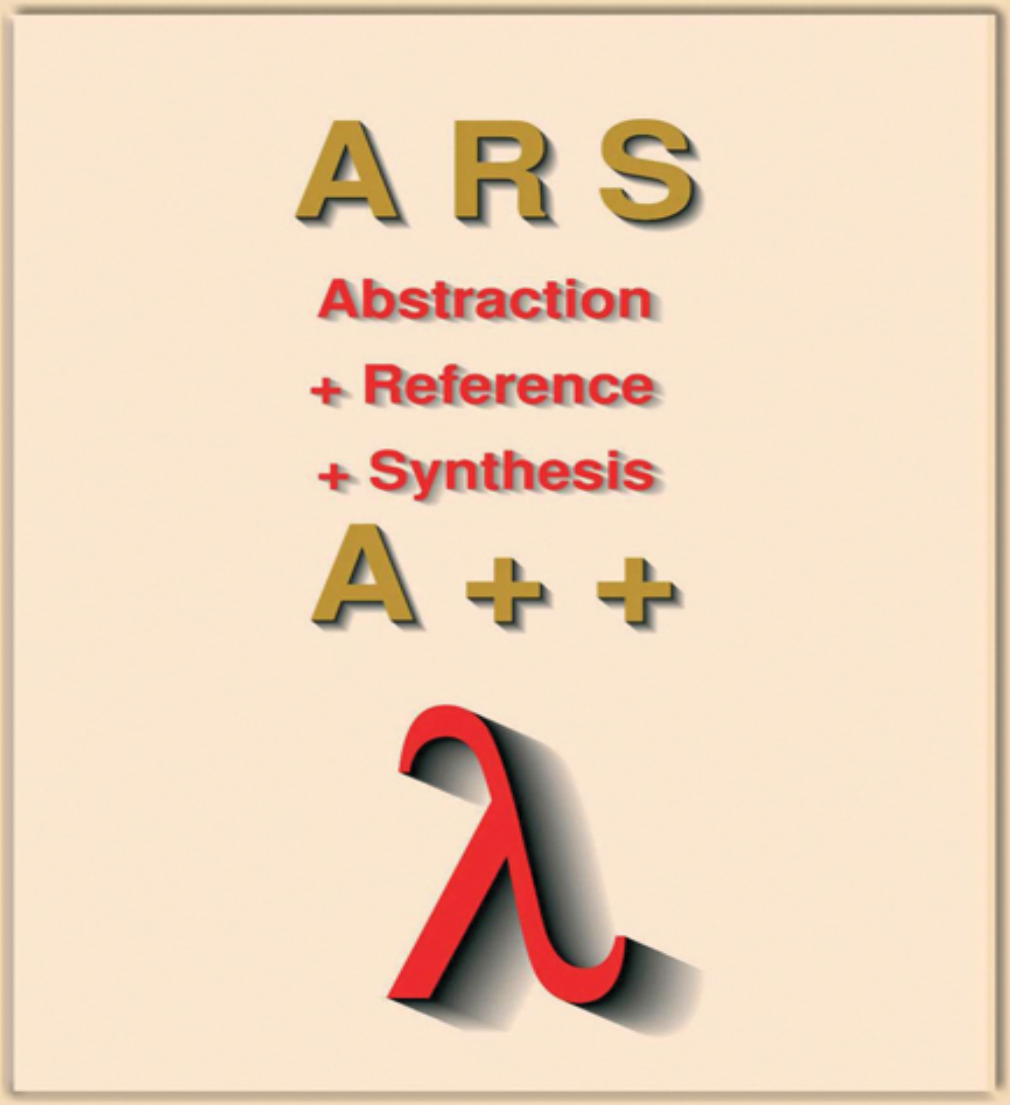
Georg P. Loczewski

# A++ The Smallest Programming Language in the World

## An Educational Programming Language




tredition®

# A++

## The Smallest Programming Language in the World

**Georg P. Loczewski**


# A++

## The Smallest Programming Language in the World

*An Educational Programming Language*

Including an Introduction to the Lambda Calculus

tredition®
www.tredition.de

IMPRESSUM



The book was set by the author using the LaTeX typesetting system and was printed and bound in the Federal Republic of Germany.

1st. Edition 2005
S.Toeche-Mittler Verlag, Darmstadt

2nd. augmented Edition 2018
tredition GmbH, Hamburg

ISBN
978-3-7469-3021-3 (Paperback)
978-3-7469-3022-0 (Hardcover)
978-3-7469-3023-7 (e-Book)

See also *A++ Die kleinste Programmiersprache der Welt*[28]



The book was set by the author using the LaTeX typesetting system and was printed and bound in the Federal Republic of Germany.

*To my wife Ursula and my sons Thomas and Johannes dedicated in love.*

# Preface

## Origin of A++

**A++** was developed in the year 2002 in the context of writing the book 'Programmierung pur' which has been published in April 2003 under the ISBN 3-87820-108-7.

The *roots of A++* lie back many years however, when I discovered Scheme which was in 1995 or to be more precise a little bit after when I read the book 'Scheme and the Art of Programming' by George Springer and Daniel P. Friedmann [32]. It was not this book that inspired me to launch the ARS or A++ project a couple of years later, it were a few statements by **Guy L. Steele Jr.**, one of the creators of the Scheme programming language, which he made in the foreword of that book.

Describing the origin of Scheme he gives homage to the *Lambda Calculus*, invented by **Alonzo Church** in the late 1930's. He talks about the importance of the principle of abstraction and makes the following statement in this context on page XV:

> Abstraction consists in treating something complex as if it were simpler, throwing away detail. In the extreme case, one treats the complex quantity as atomic, unanalyzed, primitive. *The lambda calculus carries this to a pure, unadulterated extreme*. It provides exactly

> three operations, plus a principle of startling generality.

The emphasis of the selected sentence in this quotation does not appear in the original text.

Following these words which have been quoted here, Guy L. Steele describes the three basic operations of the lambda calculus. He continues to describe the ingenuity of the lambda calculus coming up with another remarkable statement:

> Abstraction is all there is to talk about: it is both the object and the means of discussion.

Guy L. Steele's praise of the lambda calculus fascinated me, impressed me a lot and never let me loose.

There was one thing however, which bothered me: The **definition of 'abstraction'** as *'give something a name'* and the operation of abstraction in the lambda calculus do not match, at least according to my understanding. The operation of abstraction in the lambda calculus does not include a 'name giving' feature. Lambda abstractions in the lambda calculus are 'per se' anonymous. They are assigned a name only indirectly if they are passed as arguments to another lambda abstraction in the synthesis operation.

I am grateful to Guy L. Steele for letting me discover the beauty of ARS, which he describes in the mentioned text, in which he also outlines the origin of Scheme. Scheme is a programming language which has inherited a lot of the beauty and power of the lambda calculus that Guy L. Steele is talking about.

*Glenn L. Vanderburg* has used the following words to describe Scheme in his advanced text book about Java [34] on page 593:

> *Scheme might be considered the queen of programming languages: It is small, pure, and beautiful.*

Now there is another thing that bothered me though: it has to do with *how Scheme is defined in the official document*, the R5RS-Report [23]. The formal syntax definition starting on page 38 in that document does not show at all the three basic operations of the lambda calculus to be the root or the core of the scheme programming language.

The fascination of ARS, shared with Guy L. Steele, and the two things that bothered me led finally to the development of **A++** and even **ARS++**. ARS++ extends A++ into Scheme and beyond without giving up ARS as root and core of the language, showing up *'unadulterated'* in its definition[1].

Scheme is small, *A++ is even smaller*. It was a challenge for me to see how far one could go just using ARS, without creating a programming language with all kinds of whistles and bells definitely tempting a student in the process of being initiated to programming to focus his or her attention on things that are not essential to programming at all.

To develop A++ and extending it to ARS++ and to define an interface to C, C++ and Java called ARSAPI, was a great adventure and learning experience for me,[2] which I would like to share with many other people being engaged in teaching or studying the art of programming. This book covers the first part. All parts together are presented in 'Programmierung pur' ('Undiluted Programming').

## Acknowledgements

My thanks go out to many people who through their excellent work made it possible and interesting for me to get involved with programming languages in general and Scheme, the Lambda Calculus and ARS specificallly. This includes Guy L. Steele and Gerald J. Sussman, Alonzo Church, Jacques Chazarain [3], Peter Norvig [29], Samuel N. Kamin [22], Brent Benson (‘libscheme’), Richard Stallman (‘gcc’), Hans J. Boehm (‘garbage collector’) and many others.

At this point it is also appropriate to express thanks to all those people who make their software products available to all who have a need for it, software products like Linux, all GNU-licensed programs, Java, all Scheme implementations, the excellent typesetting software LaTeX with all the macro packages and again many others.

Without these people, i.e. without their books and their software, the idea to write a book on A++ and ARS++ would have never been born and the project would have never been realized.

My special thanks I want to express to my publisher Mr. Jens Toeche-Mittler, who supported the idea behind these books from the very beginning and to my family who put up with me thinking, programming and writing at times, when other tasks may have seemed and may have been more important.

Georg P. Loczewski Gross-Zimmern, September 2004

After 6 years of programming in perl following the publication of the first edition and the discovery of an

exciting lambda language, JavaScript, I thought it was time to work a little bit on A++. Douglas Crockford convincingly shows in his book 'JavaScript – The Good Parts' ([11]) that JavaScript is a beautiful and powerful language despite a few 'bad parts'. Unfortunately it didn't make in the competition with Java, back in history, because of political management decisions.

A good part of JavaScript is without any question JSON, the JavaScript Object Notation, becoming more popular now as a replacement of XML. The simplicity and expressiveness of JSON make it a good candidate as a generic vehicle for data interchange. I thought it might be interesting to merge the simplicity of A++ in respect to the internal architecture of programs with the simplicity of JSON in regard to the handling of data.

On the other hand it is always worthwhile to think about possibilities to improve something that is 5 years old. These are the reasons, A++ - The Smallest Programming Language in World, appears in its second edition.

Georg P. Loczewski Gross-Zimmern, July 2010

---

[1]The name ARS++ is derived form *ARS + Scheme + Extensions*. This means that the primitive operations of Scheme and some more have been made available to A++ in the form of predefined values and abstractions.

[2]-unfortunately a little bit late though after having been working in the field of programming for over 30 years-

# Contents

# Part I

# A++ – The Language

## Chapter 1

# Educational Programming Languages

## 1.1 Introduction

An *educational programming language* is a programming language that is designed primarily as a **learning instrument** and not so much as a tool for writing real world application programs.
In this sense *A++, Pascal, Scheme and Logo* may be considered to belong to this category of programming languages.

## 1.2 Pascal

**Pascal** has been *traditionally* used in many schools, colleges and universities in computer science classes to teach students the fundamentals of programming.
The great success of Pascal in the field of Computer Science Education results from its perfectly structured architecture forcing students to rigorously follow the rules of Structured Programming. In the 1980's, the time period, in which spaghetti-code and 'GOTO-Statements' were discovered to be the source of all evil in programming[1] Pascal was cheerfully received as a medicine against the poison that had infiltrated the programming practices of that time.

## 1.3 Scheme

More and more computer science teachers today prefer **Scheme** as the programming language of choice whenever students have to be introduced to the world of computer programming.
They argue that learning Pascal requires students to spend too much brainpower on the *syntax of a language* than on the *essentials of programming* .
They want to *protect students from losing most of the fun in programming* by getting too deeply involved in secondary things like the syntax and rules of a specific programming language.
They also believe that *Pascal* as a programming language is *less expressive than Scheme ,* imposing on students *too many limits* thus demanding from them to spend much of their intellectual energy coping with the idiosyncracies of a language instead of letting them concentrate on the solution of a given problem.
The use of Scheme on the contrary will **liberate students** initially from lower level thinking that is influenced by the implementation of a programming language and will take them up to the heights of *thinking in terms of higher levels of abstraction* making programming *simpler, more comprehensive and more powerful*.

## 1.4 Logo

**Logo** is a language especially designed to *introduce children to programming* . A device called ’turtle’ is used to make programming for children very attractive. It is amazing how fast children learn to program the ’turtle’ to draw all kinds of pictures on the screen, starting with simple lines and later the fanciest pictures after having been familiarized with basic programming constructs.

Logo is not a ’dumb’ language however that can only be used to draw lines. It is very similiar to Scheme in its expressiveness and power and can there be used by experienced programmers as well to write complex application programs. Logo is especially suited for applications in the field of *symbolic programming* and *artificial intelligence* .

## 1.5 A++

**A++** in particular is a programming language designed to provide a tool for basic training in programming*enforcing a rigorous confrontation* with the **essentials of programming** . **Programming in A++** *students learn*

- that programming problems can be solved using the powerful patterns derived from ARS (Abstraction, Reference and Synthesis)
- and that neither the knowledge of the syntax of a programming language
- nor the familiarity with all the primitive functions of a language implementation makes up the art of programming.

An *interpreter* is available in Scheme, Java, C, C++ and Python offering an ideal environment for basic training in programming, enforcing a rigorous confrontation with the fundamentals of programming languages.
In *none of the programming languages* used traditionally in introductory programming classes students are forced as rigorously as in A++ to come to a deep understanding of the essentials of programming.
This *rigorous approach* has the advantage, that students become thoroughly familiar with powerful programming patterns very fast, enabling them to learn and productively

apply the popular programming languages of the real world in a very short time.

---

[1] This era began with Prof. Dijkstra’s famous article in the Communications of the ACM with the title ’A GOTO-Statement Considered Harmful’.

## Chapter 2

# Introduction to A++

## 2.1 Purpose of A++ and Origin

### 2.1.1 Purpose

**A++** is a *minimal programming language* that has been built on the **Lambda Calculus** with the purpose to serve as a learning instrument rather than as a programming language used to solve practical problems.
A++ is introduced as a **universal learning tool for programming**, confronting students with the essence of programming and helping to master this confrontation.
It is also supposed to help become thoroughly familiar with programming patterns that can be applied in other languages needed to face the real world. Learning of new programming languages will be a lot easier and need less time after an intensive training in A++, leading to earlier productivity in the new programming language.

### 2.1.2 Origin

A++ is a language that owes it's existence to a *generalization of the base operations of the Lambda Calculus* and may very well be called the *smallest programming language in the world*.
**A++** has been developed in 2002 in the context of 'ARS based programming' covered in detail in the book 'Programmierung pur' (ISBN 3-87820-108-7) with the

purpose to serve as a *learning instrument* rather than as a programming language used to solve practical problems.

### 2.1.3 ARS — *Generalization of the Lambda-Calculus*


The pure **Lambda Calculus** is applicable only to functional programming. **A++** however is built on **ARS** which stands for the basic operations of the Lambda-Calculus in a generalized form. Guy L. Steele, one of the fathers of the Scheme Programming Language, praises the beauty of ARS in his foreword to [32] on page XV and XVI. The following phrase puts everything to a point:

> *Abstraction is all there is to talk about: it is both the object and the means of discussion.*
>
> **Guy L. Steele Jr.**

**ARS** provides a base for imperative programming and object-oriented programming as well and can be applied to programming in almost any programming language.
**ARS**, the basic operations of the Lambda Calculus *in their generalized form* are defined as follows:

> **Abstraction:** Give something a name.
>
> **Reference:** Reference an abstraction by name.

**Synthesis:** Combine two or more abstractions to create something new.

These operations may sound rather trivial and abstract but taken as principles of programming they change the style and method of programming thoroughly.
The **generalization of the Lambda Calculus** consists in defining the concept of abstraction simply by 'give something a name'. The name hides all the details of the defined. Abstraction thus defined requires an *explicit definition of a name*.
The Lambda Calculus does not allow for an explicit definition of a name. The only possibility to associate a name to a value in the Lambda Calculus is by calling a function with an argument. This operation corresponds to the synthesis operation however and not to the creation of an abstraction. Lambda-abstractions in the Lambda Calculus are 'per se' anonymous.

### 2.1.4 Name of the language


The name **A++** stands for *abstraction plus reference plus synthesis* .
The three elements of the name designate the primitive operations of the language used to build everything else.