Turing project: ALGOL 60 Tutorial

Viktor Gsteiger
University of Basel
Matriculation Number: 18-054-700

November 21, 2020
Seminar: 58826-01 - Turing Award Winners and Their Contributions

Abstract

The difficulty of learning a new programming language is inherently great. One may have no previous experience all together, one may have some experience but with another language or one may have some knowledge about the language at hand but may have forgotten large parts of the learn things again. The difficulty of learning a programming language that is not used any more and never had great commercial success is even greater, however, in the case of ALGOL 60, I am convinced, that the effort is not without benefits. ALGOL 60 is one of the grandparents of most modern programming languages and thus a direct predecessor of the tools we use every day. It is thus important to study the roots of our tools, to learn from past experiences and correct past mistakes.

Contents

1 Introduction

This tutorial aims to give the reader an introduction into the ALGOL 60 programming language. The reader should be able to program small to mid size procedures after reading this tutorial and should be able to translate and execute the ALGOL 60 program with the help of the marst translator. This tutorial does not aim to be complete and due to the inherent difficulty of learning a programming language it does not aim to lead to success.

2 Background

ALGOL 60 was the direct successor of the International Algebraic Language (IAL or later called ALGOL 58) and was a joint effort of European as well as American computer scientists in the years 1958 to 1960. With the help of the ALGOL Bulletin, a publication edited by Peter Naur, and several conferences the ALGOL 60 report could be published in 1960. ALGOL 60 did not have great commercial success on its own, however, the concepts introduced by the language can be witnessed in programming languages until nowadays.

The GNU marst translator translates programs written in ALGOL 60 into the ANSI C 89 programming language. It is part of the GNU project and currently maintained by Andrew Makhorin and the last release dates back to 2013.

3 ALGOL 60 Environment Setup

Before we can start programming in ALGOL 6, we will need to install some prerequisites to edit, translate and execute ALGOL 60 programs.

3.1 Text Editor

To edit any kind of text document, one will need a text editor. Examples include Windows Notepad, vim, EMACS, Atom or similar text editors.

The files created with the text editor are source files with ALGOL 60 programs usually having the extension ".alg".

3.2 The C Compiler

The C Compiler translates the human readable source code into executable machine language. In the case of writing ALGOL 60 programs the C Compiler is not directly accessed by the user but rather compiles the translated ALGOL 60 program into machine code.

The C Compiler usually used is the GNU C/C++ compiler. In the following subsection I will discuss on how to install the C compiler on the UNIX based Operating Systems. It is sadly not possible for me to install it on Microsoft Windows and thus I will focus on the UNIX based OS.

3.3 C Compiler Installation

3.3.1 Installation on UNIX

The GNU C/C++ compiler is mostly already installed on UNIX systems. To check whether the compiler is already install type the following into the command line:

```
$ gcc -v
```

If the GNU compiler is already installed then something like the following will be printed out to the command line:

```
Using built-in specs.
Target: i386-redhat-linux
Configured with: ../configure --prefix=/usr .......
Thread model: posix
gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)
```

If the GNU compiler is not installed on your UNIX system you will need to install it from an official GNU distribution. See the documentation on the download page for reference.

3.3.2 Installation on Mac OS

The easiest way to install the GNU compiler on a Mac OS X is to install the Xcode development environment provided by Apple. See the documentation on the download page for reference.

This tutorial has been written based on Mac OS and the examples have been translated and compiled on Catalina.

3.4 ALGOL 60 Translator Installation

The marst ALGOL 60 translator can be downloaded from any gnu mirror under `/gnu/marst/`. We will be using version 2.7 or marst released in 2013. Download the tar directory and uncompress it.

To install marst on your OS type the following into the command line at the location of the `marst-2.7` directory:

```
$ ./configure; make; make install
```

This should configure, build, and install the marst package. For more information see the `README` or the `INSTALL` file.

4 ALGOL 60 Program Structure

Before we introduce the building blocks involved in developing an ALGOL 60 program we will introduce an example ALGOL 60 program and its structure so that we may use it again for reference in the following sections.

4.1 ALGOL 60 Hello World Example

An ALGOL 60 program can be constructed with the following parts:

- Procedures
- Variables
- Statements
- Comments

A simple example to display various parts of an ALGOL 60 program would be the following:

```
procedure main();
        comment a first ALGOL 60 program
        begin
                outstring(1, "Hello, world!\n")
        end
end main;

main();
```

The parts of the above program are the following:

1. The first line declares the procedure which we called the main procedure. The name of the procedure can be changed.
2. The next line is a comment which will be ignored by the translator and is used to comment on the code at hand to make it easier for fellow programmers to understand the intention of the program.
3. The `begin` keyword signifies that a block of the procedure starts here.
4. Following comes an output keyword `outstring` which displays the string given to the first output channel.
5. The last line calls the main procedure and executes it with it.

4.2 Compile and Execute an ALGOL 60 Program

We will now save the ALGOL 60 program, translate it, compile it, and run it. The steps to do this are the following:

1. Open your text editor and type in the above program.
2. Save the file as `hello.alg`.
3. Open a command line and navigate to the directory where the above program has been saved.

4. Type `marst hello.alg -o hello.c`.
5. If there are no errors the translator creates the C file `hello.c`.
6. Compile and link the file with the following command `gcc hello.c -lalgol -lm -o hello`.
7. Run your executable `./hello`.
8. If everything worked fine you should see `"Hello, world!"` printed on the command line.

5 ALGOL 60 Basic Syntax

5.1 Formal Notation

The notation used in this part of the tutorial is the Backus Naur form also used in the original AGLOL 60 report. The notation is best explained with an example:

$$< \text{ab} >::=(| < \text{ab} > (|\text{example}|) \tag{1}$$

Where the sequence of character enclosed in brackets represent meta-linguistic variables which are represented by a sequence of symbols. The ::= and | signify meta-linguistic connectives. | has the meaning or. Any symbol in a formula that is neither a variable nor a connective denotes itself. Variables can be replaced with their own definition. So the example signifies a recursive rule for the formation of values of the variable <ab>. Some values for <ab> are:
example(or)((

5.2 Symbols

The basic symbols of the ALGOL 60 programming language consists of letters, digits, logical values, and delimiters. With the basic symbols of ALGOL 60 every ALGOL 60 program can be created. The basic symbols themselves have no semantic values and are combined together to create every program. The basic symbols are made up as follows:

5.2.1 Letters

$$\begin{aligned} < \text{digit} >::=& a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z| \\ & A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z \end{aligned} \tag{2}$$

Which are used for forming identifiers and strings.

5.2.2 Digits

$$< \text{digit} >::=0|1|2|3|4|5|6|7|8|9 \tag{3}$$

Which are used for forming numbers, identifiers and strings.

5.2.3 Logical values

$$< \text{logical values} >::=\textbf{true}|\textbf{false} \tag{4}$$

Which have a set meaning as boolean logical values.

5.2.4 Delimiters

$$\begin{aligned}
< \text{delimiter} >::= & < \text{operator} > | < \text{separator} > | < \text{bracket} > | < \text{declarator} > | \\
& < \text{specificator} > \\
< \text{operator} >::= & < \text{arithmetic operator} > | < \text{relational operator} > | \\
& < \text{logical operator} > | < \text{sequential operator} > \\
< \text{arithmetic operator} >::= & + | - | * | / | \% | \hat{} | ** \\
< \text{relational operator} >::= & < | <= | = | >= | > | != \\
< \text{logical operator} >::= & == | -> (\text{meaning } \supset) | | (\text{meaning or}) | \& | ! \\
< \text{sequential operator} >::= & \textbf{go to} | \textbf{if} | \textbf{then} | \textbf{else} | \textbf{for} | \textbf{do} \\
< \text{separator} >::= & , | . | \# | : | ; | := | \textbf{step} | \textbf{until} | \textbf{while} | \textbf{comment} \\
< \text{bracket} >::= & (|) | [|] | " | \textbf{begin} | \textbf{end} \\
< \text{declarator} >::= & \textbf{own} | \textbf{Boolean} | \textbf{integer} | \textbf{real} | \textbf{array} | \textbf{switch} | \textbf{procedure} \\
< \text{specificator} >::= & \textbf{string} | \textbf{label} | \textbf{value}
\end{aligned} \tag{5}$$

All delimiters have a fixed meaning for which is mostly obvious, or else the meaning will be explained at the appropriate section.
Typographical features such as blank spaces can be inserted between symbols, however, multi-character symbols should contain no blank space.
The separator **comment** has special importance and has the purpose of writing symbols into the code that will not be translated into machine executable code. The commenting symbols are all symbols between the separator **comment** and the separator ;.

5.3 Identifiers

In ALGOL 60 an identifier has no inherent meaning, but serve the identification of variables, arrays, labels, switches and procedures. An identifier starts with a letter and is followed by zero or more letters or digits. Every combination is allowed except the previously defined delimiters. Some examples for identifiers are as follows:

Soup
V17a
MARILYN

5.4 Numbers

Numbers are used for arithmetic operations. There are two types of numbers in the ALGOL 60 programming language. Both have the same functionalities, however different ranges of size. The types of numbers are **integer** and **real**. Integers are any positive or negative combination of digits. Reals are any integer including a decimal fraction and/or an exponent part (denoted by the symbol #). More on this in the section ALGOL 60 Data Types. Some examples for numbers are as follows:

```
0
.5384
-.083#-02
```

5.5 Strings

A string is any sequence of basic symbols not containing ". Strings are used as actual parameters of procedure, for example also the procedure `outstring` which has been included into the translation program for output functionalities. Due to the translation to a C program the strings may be coded as usual in C fashion. Escape sequences like \n are allowed. To use double quote in a string use a backslash \". Some examples for strings are as follows:

```
"This is a string"
"This is another \"string\""
```

6 ALGOL 60 Data Types

As the original ALGOL 60 document was written without a specific hardware implementation in mind the ALGOL 60 data types used in this tutorial will reflect the C language data types in sizes. Data types define how much space will be occupied in storage.

6.1 Integer Types

As mentioned in subsection Numbers integers refer to any positive or negative concatenation of numbers without any exponent or decimal fractions. The values possible are between `-32'768` to `32'768` if int is stored in 2 bytes or between `-2,147,483,648` to `2,147,483,647` if int is stored in 4 bytes. To get the exact possible size of the integer write a short program as follows:

```
inline("printf(sizeof(int));")
```

The inline procedure is a pseudo procedure implemented by the MARST developers to signify what code will be one to one translated into C code.

6.2 Real Types

Again, as mentioned in subsection Numbers reals refer to any positive concatenation of digits which may include exponents and/or decimal fractions. The values possible are between `1.2E-38` to `3.4E+38` with a precision to 6 decimal places if float is stored in 4 bytes

7 ALGOL 60 Expressions

The main constituents of any ALGOL 60 programs describing algorithmic processes are arithmetic, Boolean, and designational, expressions. These expressions contain logical values, numbers, variables, function designators, and operators.

7.1 Variables

A variable represents a single values. This value can be used in expressions and can be used to form other values and may be changed by means of assignment statements. Array expressions with their identifiers are also considered variables. The identifiers of arrays are enclosed in subscript brackets []. The type of a variable is defined in the declaration of the variable itself (see Type Declaration for more information on types) or for the array identifier (see Array Declaration for more information on arrays). Examples for variables are as follows:

```
beta
Q[7, 2]
a17
```

7.2 Function designators

A function designator represents a single value that can be attained by the application of a given set of rules defined by a procedure declaration (see Procedure Declaration for more information on procedures) to a defined set of parameters which can either be single values or variables. Examples for function desigantors are as follows:

```
Compile("Test") Stack:(P)
J(1 + s, n)
Rhesus
```

7.3 Arithmetic expressions

Arithmetic expressions, as the name says, are rules for computing numerical values. Simple arithmetic expressions are the application of the arithmetic operations of the rule upon the actual numerical values of the primaries involved in the expression. The numerical values of primaries are either simply the values given in the case of numbers or in the case of variables the currently assigned values of the variables (see Assignment Statement for more information on assignments). For functional designators it is the value received by executing the corresponding procedure. The normal arithmetic rules apply. Thus also the precedence from left to right with the exponent operator having the highest precedence, the multiplication or division operators the second highest and the addition or subtraction operator the third highest precedence. Expressions within parentheses are evaluated on their own and further calculated in the subsequent calculations. Examples for simple arithmetic expressions are as follows:

```
w * u - Q(S + C) ^** 2
a * sin(omega * t)
```

There is also the possibility of more complex arithmetic expressions which involve Boolean expressions. In this case the value of the arithmetic expression is calculated from the Boolean expressions that are true. Examples for complex arithmetic expressions are as follows:

```
if q > 0 then U + V else if a * b > 17 then U / V else 0
if s then n - 1 else n
```

7.3.1 Arithmetic expression types

The types of arithmetic expressions must be integer or real (see ALGOL 60 Data Types for more information on types).
The operators +, - and * have the conventional meaning and will return integer if all operands are integer, else real.
The operation <term> / <factor> and <term> % <factor> both denote division while / is defined for all four combinations of integer and real and will give a result of type real while % is only defined for two operands of type integer and will return an integer. / is defined as the multiplication of the term with the reciprocal of the factor. % is defined as the multiplication of the sign of the / division of the two operands with the largest absolute integer that is smaller than the / division of the two operands.
The operation <factor>^** <primary> means exponential, where the factor is the base and the primary the exponent.

7.4 Boolean expressions

Boolean expressions are rules for calculating logical values where the principles of evaluations are analogous to the principles of arithmetic evaluations. The operators of Boolean expressions are defined as follows:

b1	false	false	true	true
b2	false	true	false	true
!b1	true	true	false	false
b1&b2	false	false	false	true
b1|b2	false	true	true	true
b1->b2	true	true	false	true
b1==b2	true	false	false	true

The Boolean expressions <, <=, =, >, >=, and != are applied on two arithmetic expressions.

8 ALGOL 60 Statements

8.1 Assignment Statement

8.2 Go To Statement

8.3 Conditional Statement

8.4 For Statement

8.5 Procedure Statement

9 ALGOL 60 Declarations

9.1 Type Declaration

9.2 Array Declaration

9.3 Switch Declaration

9.4 Procedure Declaration

9.5 Variable Declaration

10 Procedures

11 ALGOL 60 Scope Rules

12 Simple Programs

13 Recursion