Plankalkül

The Plankalkül (*calculus of programs*) was the first high-level programming language conceived in the world. It was designed by Konrad Zuse, the German inventor, between 1943 and 1945, that is, at a time when the first computers were being built in the USA, UK and Germany. It represents one of the major contributions to the history of ideas in the computer field, although it was never implemented for any kind of machine.

The Plankalkül corresponds to Zuse's mature conception of how to build a computer and how to allocate the total computing work to the hardware and software of a machine. Zuse called the first computers he constructed (the Z1, Z2, Z3 and Z4) "algebraic machines" in contrast to "logistic machines". The first were specially built to handle scientific computations, the latter could deal with scientific but also with symbolic processing. Zuse's "logistic machine" was never built, but its design called for a one-bit word memory and a processor which could only compute the basic logic operations AND, OR and NOT. It was a sort of minimal machine. Since the memory consisted of a long chain of bits, they could be grouped in any desired form to represent numbers, characters, arrays, etc.

The Plankalkül was the software counterpart of the logistic machine. Complex structures could be built from elementary ones, the simplest being a single bit. Also, sequences of instructions could be grouped into subroutines and functions, so that the user had only to deal with a very abstract instruction set that masked the complexity of the underlying hardware. The Plankalkül exploited the concept of modularity, so important today in computer science, almost in an extremist way: several layers of software make the hardware transparent for the programmer. The hardware itself is able only to execute the absolutely minimal instruction set.

In Plankalkül, the programmer uses variables to perform computations. The notation is such that intermediate results are labeled Z1, Z2, Z3, etc. Input variables are labeled V1, V2, V3, etc., and results are labeled R1, R2, R3, etc. To describe a variable and its type, Zuse used the "row notation", shown below:

	Z
V	1
K	2
S	5.o

These four lines define the variable Z1 (note that the index is written in the next line, the "V" line), with "structure" 5.o, that is, five times the structure "o", which represents a single bit. The K-line tells us which component is being refereed to. In this case we refer to the second bit of the five bit field Z1. Therefore, the notation is two-dimensional, although it could be compressed on a single line. In a modern programming language, we would write Z1[2].
There are no separate variable declarations; any variable can be used in any part of the program and its type is written together with the name.

The type of a variable could be selected in a very flexible way. The only primitive type was "o" (a bit). A group of *n* bits was denoted as *n*.o. A group of *m n*-bit numbers, as *m*.*n*.o, and so forth. Any kind of primitive data type (characters, integers, reals) as well as vectors and

matrices can be defined in this way. A type could be abbreviated using another letter, and this letter could be used as building block for another composite type.

Variable assignment is done as in modern programming languages. The new value overwrites the old value of a variable. There are several operations which are also used like in other programming languages (addition, subtraction, etc.). The addition of two variables V1 and V2 (eight bits each) can be stored in an intermediate variable Z1 using the following piece of code:

	V	+	V	⇒	Z
V	1		2		1
K	1		3		1
S	5.8.o		5.8.0		5.8.0

In Pascal, we would just write Z1[1]:=V1[1]+V2[3]. Note that the variables V1, V2 and Z1 have the same type: an array of five numbers of eight bits. The programmer has to see to it himself that the assignments refer to variables of the same type, since there is no type checking.

Arrays of objects can be indexed by using an auxiliary variable. The use of the index variables is shown using a line, as shown below:

	V	V	⇒	Z
V	1	2		1
K		2		1
S	5.8.o	5.8.0		5.8.0

In this example, the second component of the array V2 contains the index for the array V1. The number is copied to the fist component of Z1. In Pascal we would write: Z1[1]:=V1[V2[2]].

Boolean operations produce results which are single bits. The zero is interpreted as FALSE and the 1 as TRUE. Boolean results can be used in conditional instructions.

Plankalkül can deal with conditional instructions of the "IF-THEN-ELSE" type. They are written as guarded instructions of the form: A → B. If the guard A is true, the command B is executed.

Blocks of instructions can be written in the Plankalkül by separating each instruction with a vertical line or by writing the instructions one under the other. A block is enclosed in parentheses. A block counts later as a single instruction and can be made part of another block.

There is also an iterative operator W, which repeats the execution of a sequence of instructions until all guards in the body of the loop fail:

$$W \left[\begin{array}{l} A \rightarrow B \\ C \rightarrow D \\ E \rightarrow F \end{array} \right]$$

Here, the scope of the W covers the three guarded instructions, which form a block. The loop is repeated if any of the guards are true. Execution of the loop is terminated when the three guards A,C, and E fail within the same iteration.

The elementary Boolean and arithmetic operations, guarded commands and the W control structure form the basis of the Plankalkül. Other control structures and commands can be built using them. There is for example a W1 control structure that would correspond to the FOR command in a modern programming language, that is, an iteration that is performed a certain number of times. There are also other more specialized constructions that employ quantors ("there exists an x such that", "for all x", etc.) but they can be expressed also using the basic elements mentioned above. Zuse never built a compiler or interpreter for the Plankalkül, but it seems that he was well aware that the more complex portions of the Plankalkül could be written using the basic commands.

Subroutines and functions could be written in Plankalkül. A declaration was put in front of the code to make it clear which variables were the arguments and which the results. This declaration was the "boundary portion" (*Randauszug*) of the procedure. It was possible to give also operators as arguments. A subroutine could be written, for example, that received as argument the operator "+" or the operator "×", so that the same general code could be compiled with a different operator in the body of the routine. One complication of this scheme was the absence of a clear distinction between local and global variables. Most of Zuse's draft of 1945 deals only with global variables, but he also indicates that variables in different programs can have the same name, but refer to different memory localities. Subroutines could be used also as functions: Kla(*x*), in an example given by Zuse, is a function that checks if a character *x* is an opening parentheses and returns a Boolean value.

Although Zuse published some small papers about the Plankalkül and tried to make it known in Germany, the language fell into oblivion. The main problem was its ambitious scope, the large variety of instructions that it contained, its modular architecture which called for incremental compilation, and the availability of dynamical structures and functionals. Also, some aspects of the semantics are not quite clear and the absence of type checking would have made it extremely difficult to debug. A practical implementation of the Plankalkül would certainly require a major revision of Zuse's draft of 1945. However, Plankalkül was way ahead of its time and many of the concepts in which it was based were only rediscovered much later. In the case of the Plankalkül, Konrad Zuse suffered the same fate as Charles Babbage with the Analytical Engine. Babbage had the right concepts but the wrong hardware. After 1945, many more years would be needed until programming languages could achieve the level of sophistication of Plankalkül.

References

Zuse, Konrad. *Der Plankalkül*. Bonn: Technical report 63, Gesellschaft für Mathematik und Datenverarbeitung, 1972.

-Raúl Rojas