Hochschule für Technik Stuttgart

Concepts of Programming Languages

2nd Week

Evolution of Early Programming Languages

Prof. Dr. Peter Heusch

Historical Programming Languages

Historically, programming was not done with formal languages but other techniques:

- Mechanical Switching Devices (Z1, Z2)
- Plugboards (ENIAC, IBM 604)
- Punched cards (Jaquard, Mark I)

Theorists where interested in powerful mechanisms, not syntactic elegance

During the first 20 years, efficiency was preferred to elegance

The Plankalkül – 1st higher PL

Konrad Zuse (1910-1995) was a German computer pioneer

Zuse invented the Plankalkül in 1946:

```
P2 max (V0[:8.0],V1[:8.0]) =>
     R0[:8.0]
V0[:8.0] => Z1[:8.0]
   (Z1[:8.0] < V1[:8.0])
     -> V1[:8.0] => Z1[:8.0]
   Z1[:8.0] => R0[:8.0]
   END
```

First implementation in the 70s, with modified syntax

http://de.wikipedia.org/wiki/Konrad_Zuse

Features of Plankalkül

The Plankalkül supported many modern programming languages features:

- Alternatives & Loops
- Arrays & Structures
- Functions

Notation was not adapted to contemporary computers and not constructed for compilers:

- Originally notation was two-dimensional
- Numbers had arbitrary precision
- Plankalkül was not written for real computers
- First Implementation during the 70s

Programming in the 40s & 50s

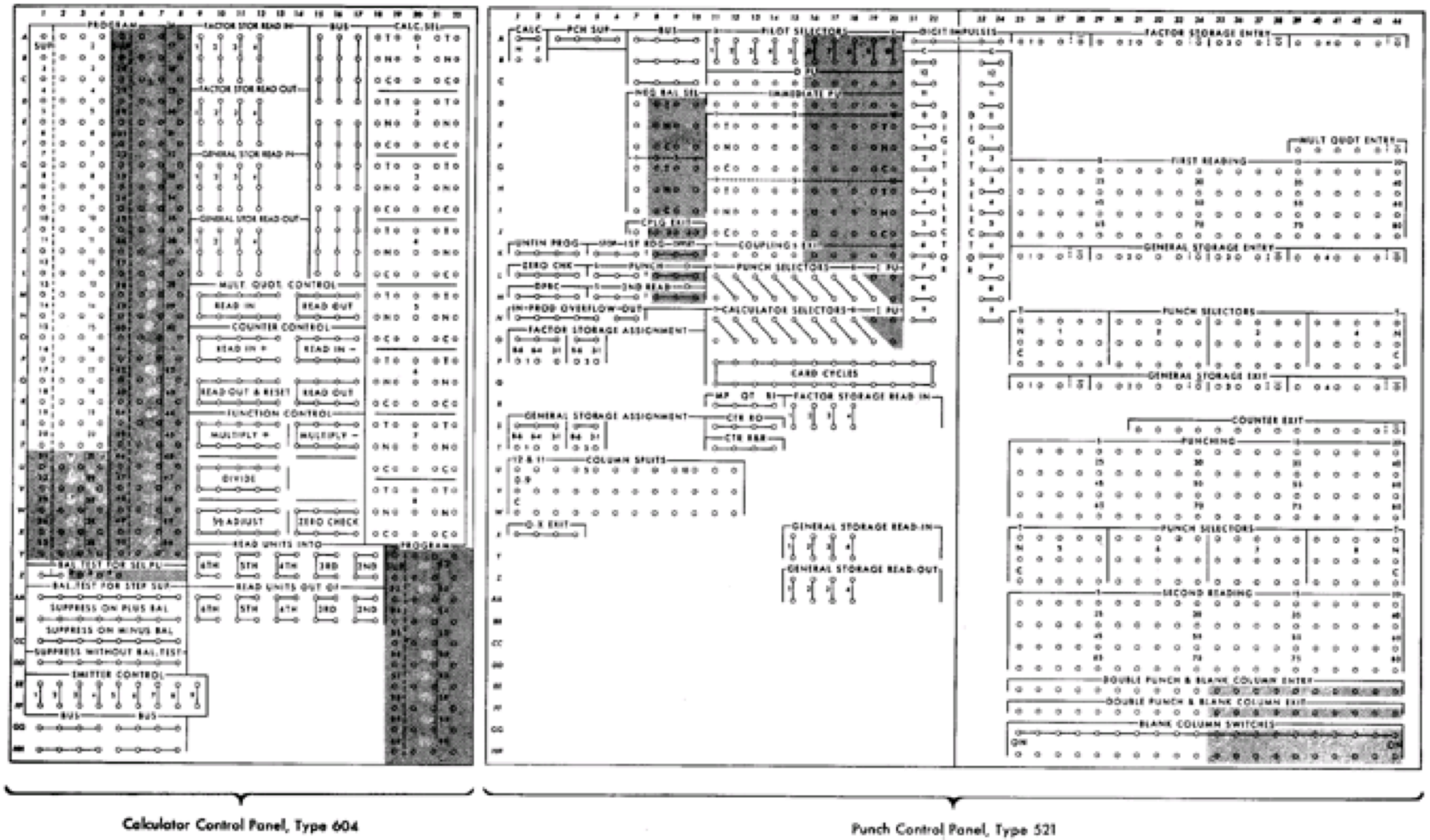

FIGURE 4. CONTROL PANELS

The shaded hubs represent additional capacity.

IBM 604 control panels, http://www.columbia.edu/acis/history/604-board.html

Accounting Machines

Typically machines were built to perform accounting procedures

At this time, programming meant performing statistical computations:

- If value in columns x to y is z then add values to register r1, else add to register r2
- If value in columns x to y changes, then output register r4 and reset it

Looping was done implicitly by repeating the program for every input card

The Pre-Von-Neumann Machines

Programs were written on punched tapes

Jumping was performed by skipping the given amount of steps forward (mechanical jump)

Advantage:

- Program storage was cheap (paper tape)
- Theoretically, it was possible to transform source code to machine code

Disadvantage:

- Program storage was slow, especially jumping
- Programs tended to wear out (paper degradation)

The First Von-Neumann-Machines

General idea: One Storage for program code and data

Allowed for compiling source to binary code

Memory implemented as magnetic storage

Advantages:

- Program storage was very fast and quite stable

Disadvantages:

- Program storage was expensive, 100 vacuum tubes cost the average annual salary of a craftsman
- As a consequence, program storage was very small

Programming in Machine Code

Programs were coded as number sequences
Reasonably easy to learn, since only very few instructions existed:

OPERATION CODES

PROGRAMMING

THERE ARE 44 possible operation codes used in programming for the Type 650. In order to simplify the understanding of all operation codes, those operations that function in a similar manner are treated as a group. Thus, when one understands the basic function of one of the group, the others should be equally clear. The explanations are grouped as follows:

1. Read and punch
2. Transfer
3. Add and subtract without reset
4. Add and subtract with reset
5. Add and subtract absolute value
6. Multiply and divide
7. Shifting
8. Branching
9. Table lookup
10. Miscellaneous

One of the most simple problems for the Type 650 would be to read an input card, transfer some or all of the information to punching locations, and punch an output card. Reading and punching are presented first and then transfer of factors in general storage.

Machine Code and Assembler Code

No need for assembler code during the 50s:

- Machine code was easy
- Changing the machine changed the machine code

First assembler programs were invented to optimize binary programs (IBM 650)

With the IBM 1401 machine codes became too complex to memorize:

- Multiple encoding formats (2 Byte – 6 Byte)
- Too many different machine operations

The /360

The first machine that could in one machine:
- Perform business calculations
- Perform scientific calculations

Even though internally the models were completely different, the machine code was their unifying link:
- Any program ran on any model
- Machine upgrades and portability were possible

Invention of the „Software industry“: it was possible to write software for others

Assembler Languages

Assembler programs offer the full power of the machine to the programmer:

- All possible instructions of the CPU
- Access to all registers, e.g. for overflow detection
- A 1:1 correspondence between source and binary

But there are also disadvantages:

- Code is hard to read and even harder to debug
- Inserting code needs recomputation of addresses

The First Higher Languages

Even in the 50s it was obvious that machine code was a bad choice:

- Programs had to be completely rewritten when changing to another machine
- It was nearly impossible to find errors
- Binary code corrections were very hard

Main interest in the 50s was either:

- Numerical computations
- Business programming
- Symbolic computations were of theoretical interest

The Fortran Language

Idea: Write your program in a uniform notation that is translated to machine code by a compiler

The invention of a compiler needed a machine, where instructions were stored in main memory

Writing a Compiler was a very complex task, but could be kept simple by modularizing:

- Processing the input (Language depending)
- Generating the output (Machine depending)

The 1401 Fortran compiler had 63 passes

The Design of Fortran

Design of Fortran was done by John Backus „around“ the available hardware IBM 704:

- First machine with floating point arithmetic
- Special registers for indexing

Available machine features influenced the language:

- Array handling was available
- Input/Output was not provided
- String handling was also not provided

Features of Fortran I

Efficiency was thought as a major goal

Names could have up to six characters

Post-test counting loop (`DO`)

Formatted I/O

User-defined subprograms

Three-way selection statement (arithmetic `IF`)

No data typing statements, data type was determined from variable name (i, j: integer, f, g, x, y z: float)

Support for complex numbers

The Fortran I compiler

Needed 18 person years to be written
Released in April 1957
Practical programming length was limited to 400 lines of code (poor reliability)
All statements had to be in one program
Code was as good as hand optimized code
Was a success, because programmers were rare
„From now on, programs will be error free“

Sample of Fortran

C = FOR COMMENT	CONTINUATION	FORTRAN STATEMENT	IDENTIFICATION
STATEMENT NUMBER			
1 5	6	7 72	73 80
C		PROGRAM FOR FINDING THE LARGEST VALUE	
C	X	ATTAINED BY A SET OF NUMBERS	
		DIMENSION A(999)	
		FREQUENCY 30(2,1,10), 5(100)	
		READ 1, N,.(A(I), I = 1,N)	
1		FORMAT (I3/(12F6.2))	
		BIGA = A(1)	
5		DO 20 I = 2,N	
30		IF (BIGA-A(I)) 10,20,20	
10		BIGA = A(I)	
20		CONTINUE	
		PRINT 2, N, BIGA	
2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
		STOP 77777	

http://ep.yimg.com/ca/I/paulgraham_2147_2899146

The Fortran II & IV compiler

- Fortran II mainly bug fixing release of Fortran I
- Allowed for separate compilation of programs:
  - Modularization allowed for larger programs
  - Separate compilation allowed for standardization of algorithms
- Fortran IV introduced new language features:
  - Complex numbers
  - Passing subroutines names as parameter
  - Explicit type declarations

Further development of FORTRAN

Fortran IV became ANSI-Standard in 1966

Fortran 77 became Standard in 1978 with new features

Fortran 90 introduced:

- Parameter type checking
- Recursion
- Modules
- Dynamic arrays

Language is still under development, e.g. by object oriented features

COBOL (1960)

Hardware-independent language for business applications

Idea: Make using computers easy for the non-programmer:

- Very verbose syntax: MOVE a TO b
- Separation of code and data allows programs to share same data sets

Builtin features for regular tasks:

- Sorting data by defined keys
- Writing reports

A COBOL Sample

Every cobol program is split into 4 divisons:

- Identification division: The author of a program
- Environment division: The runtime environment
- Data division: All data structures
- Procedure division: The executable code

A Sample:

```
IDENTIFICATION DIVISION.
    PROGRAM-ID. HELLO-WORLD.
PROCEDURE DIVISION.
    DISPLAY 'Hello, world'.
    STOP RUN.
```

Further development of COBOL

COBOL 74 reflected developments inside the runtime environments, e.g. the appearance of databases

COBOL 85 introduced structured programming constructs like END-IF

BASIC (1964)

Idea: Construct a Language for the sole purpose of teaching students programming: Beginners' All-purpose Symbolic Instruction Code

Main goals:

- Steep learning curve
- Be everybody's darling
- Easy implementation even on small machines

Variable type determined by name

A BASIC Sample

```
10 let guess = random(100)
20 print "Please input a number between 1 and 100"
30 read a
40 if a = guess then goto 70
50 if a < guess then print "Too small";
   goto 30
60 if a > guess then print "Too large";
   goto 30
70 print "Congratulations, you got it"
```

Modern Basic dialects don't need line numbers any more

PL/I (1964)

Idea: The first universal programming language
- Replace COBOL (business)
- Replace FORTRAN (scientific)
- Replace Assembler

Designed by a dedicated design committee

Started as extension of Fortran IV

Was extended to support a large variety of features

Implementation lacked behind because tools were not available

Properties of PL/I

PL/I popularized a lot of modern programming language concepts:

- Modularization
- Exception handling
- Recursion
- Introduction of Pointer types / large variety of types and data structures
- Array sections

Still in use as SPL or PL/S inside development of Mainframe operating systems

Summary

A lot of programming languages from history are still in use today:
„I don't know what the next generation scientific language will look like, but its name will be Fortran“

The year-2000-related efforts showed how many companies still use code from the 70s:
„Last modification: 19.06.1976, adapted some new features of COBOL 74“