IBM University and Vocational Education

Programming Language C

Learning about the Procedural Programming
Structured Programming with C

BA-Stuttgart | 1-Nov-2003 | Arne Heimeshoff

Agenda

- Part I: Introduction
- Part II: Derived Data Types
- Part III:

Part I: Introduction

Historical and Technical Overview

Table of contents

- **History**
- Algorithms
- Nassi-Shneiderman-Diagrams
- Programming Tools
- Variables
- Preferences / Attributes of C
- Constant Values
- Operators and Operations
- Functions
- Classes of Data Types
- Local and Global Variables

History

- In 1967 Thompson invented the programming language B in which the operation system UNIX is implemented

- He looked for a superassembler that is able to move programs to other systems and has the following attributes:

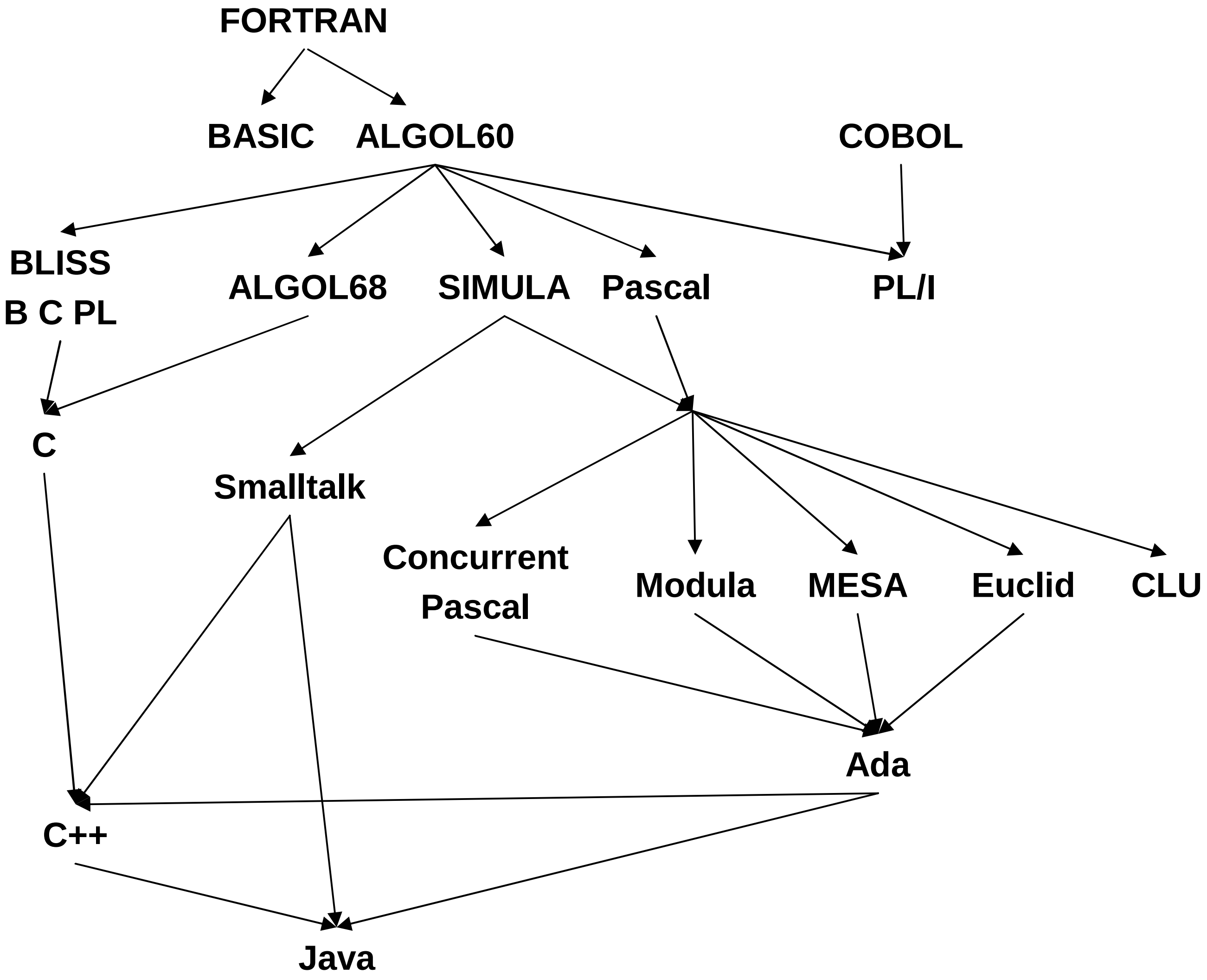
  - structured programming is possible
  - close to the hardware implementation like assembler
  - performance is nearly equal to assembler

- 1972 Ritchie (Bell Labs) worked on the programming language C, a language with a code generator and predefined data types

History

- In 1973 UNIX was implemented in C (that means 1/10 of the assembler code)
- 1978 Kerninghan and Ritchie have written a book named “The Programming Language C” what is today known as the C-Bibel
- 1989 there were several dialects unified within the ANSI C-Standard
  - libraries had been included standard, too
  - today, the ANSI standard has been replaced by the ISO standard ISO / IEC 9899

History


FORTRAN

BASIC ALGOL60 COBOL

BLISS
B C PL

ALGOL68 SIMULA Pascal PL/I

C

Smalltalk

Concurrent
Pascal

Modula MESA Euclid CLU

Ada

C++

Java

Table of contents

Algorithms

- An algorithm is a code of practice to solve a problem, similar to a cooking recipe
- An algorithm needs
  - objects
  - operations, that means operators
  - defined state at the beginning
  - defined state at the end

> *Every problem to be solved with a procedural programming language needs at least one algorithm!*

Algorithm of Euklid

The algorithm of Euklid looks for the greatest divisor in common.

- You need some object of a defined data type
  - call it: x and y
  - at the beginning each of the variables has a value
  - at the end their value is their greatest common divisor
- You will need some operations, such as

  compare

  substract

  assign

Algorithm of Euklid

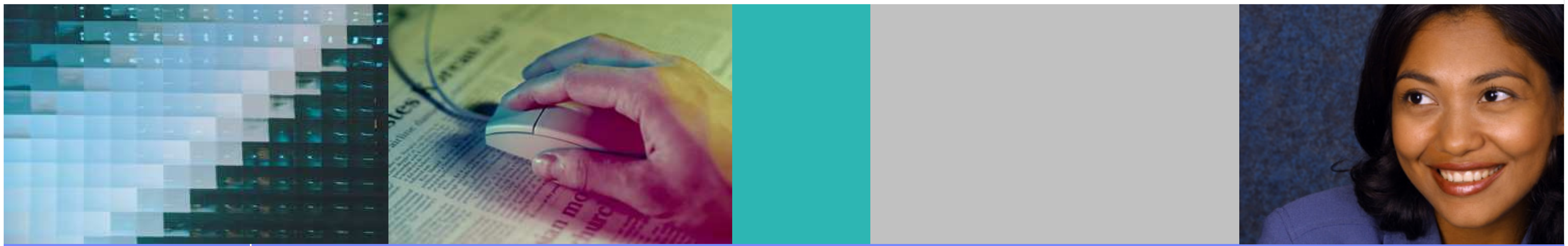
- There is a sequence of commands (operations)
- Some constructs will have influence on this sequence of commands
  - choice of alternative ways (selection)
  - repetition of instructions (iteration)
  - linear order of commands (sequence)
- You have a starting point and an end, between which you find the commands

Between the start and the end of an algorithm you have a sequence of commands.

Table of contents

Nassi-Shneiderman-Diagrams

- The order in which commands are executed is named the **control flow**
- directives which can influence the order of executing the commands are **control structures**
- The sequence of executing instructions within the program are organized by structured programming.
- To show the structure of a program use the **Nassi-Shneiderman-Diagrams**



Nassi-Shneiderman-Diagrams

- subroutines will be used several times, reusable program code
- subroutine will make a program code easier to read for other people
- go into details step by step down to the program code
- e.g. input, process, output
- with complex problems you need some iterations to come to the program code level
- normally you do not go to the program code level because then you have redundant information → not preferable

Nassi-Shneiderman-Diagrams

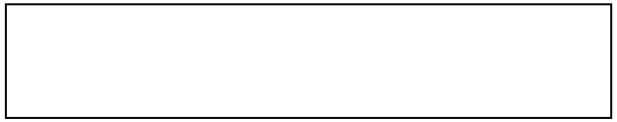
“To make the control flow of a program visible Nassi and Shneiderman have designed the structograms which are often called Nassi-Shneiderman-Diagrams. These diagrams are utilities of structured programming and don’t contain e.g. goto statements”

Nassi-Shneiderman

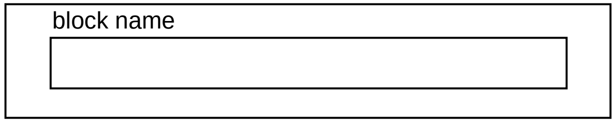
- allowed are all instruments from the structured programming
  - sequence
  - iteration
  - selection
- The flowcharts will be replaced by the Nassi-Shneiderman-Diagrams

Nassi-Shneiderman

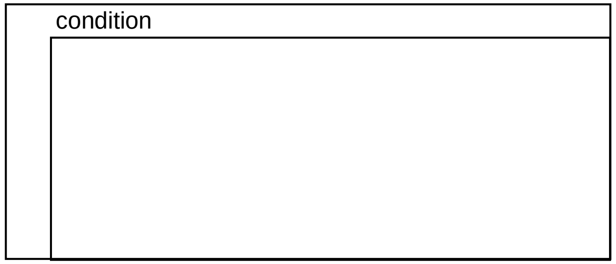
- pass, sequence

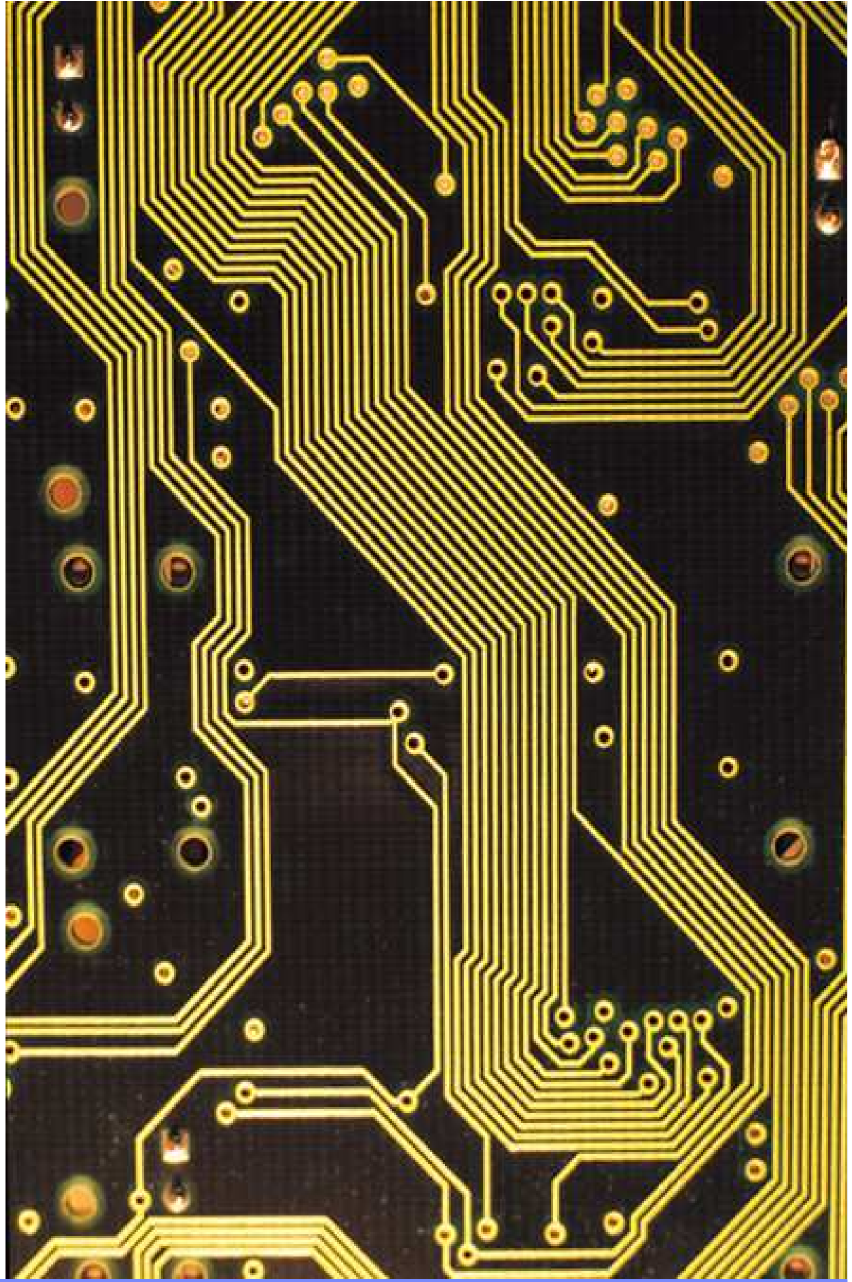


- block



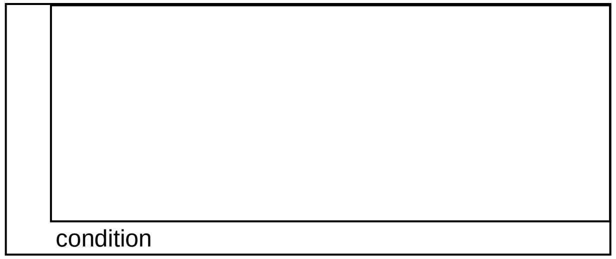
- iteration



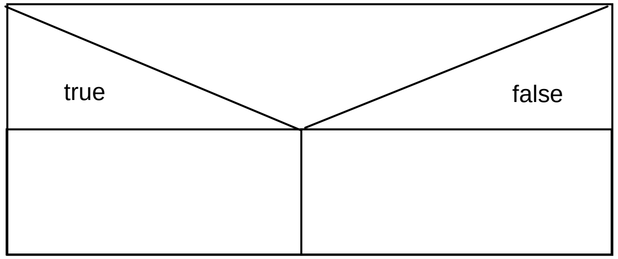


Nassi-Shneiderman

- iteration



- selection



- break

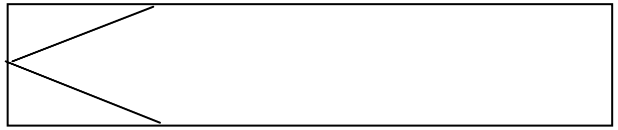


Table of contents

Programming Tools

- Preprocessor
- Compiler
  - lexical analysis
  - syntax breakdown
  - semantic interpretation
  - code generation
- linker
  - static links
  - dynamic links
- runtime environment
- loader
- debugger
- development environment

Preprocessor

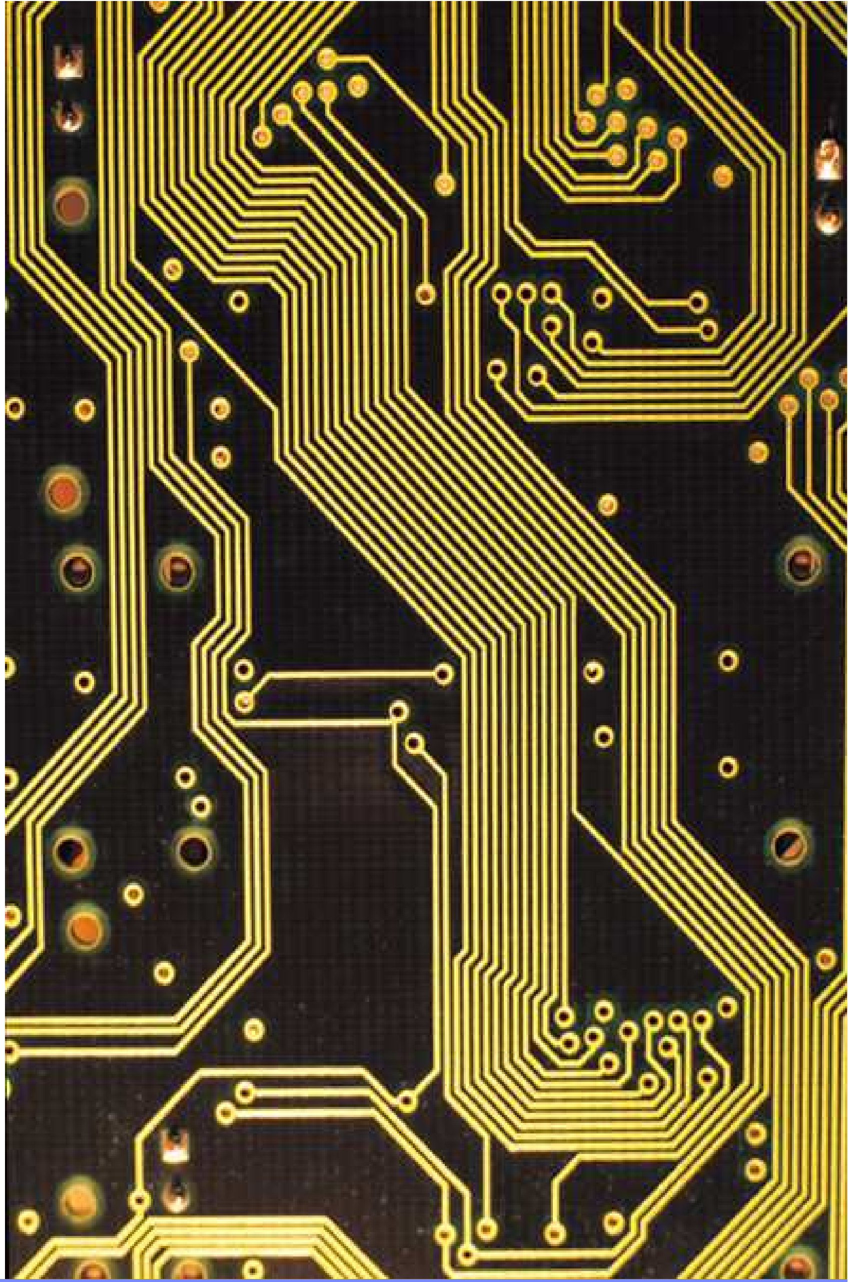
- tasks of the preprocessor
  - include files
  - substitute text
  - conditional compiling

- working structure of the preprocessor
  - join lines (*new line* and ‘\’)
  - sample the program code into tokens and space characters
  - replace comments with blanks
  - include files
  - substitute macros
  - change reserved characters (‘\...’)
  - assemble adjoining character strings

> *The preprocessor is the first working entity to become an executable program.*

Preprocessor

- inserted elements are parts of
  - type names (defined, composed)
  - data types
  - macros
  - defined constants
  - prototypes of functions
    - printf ();
    - scanf ();

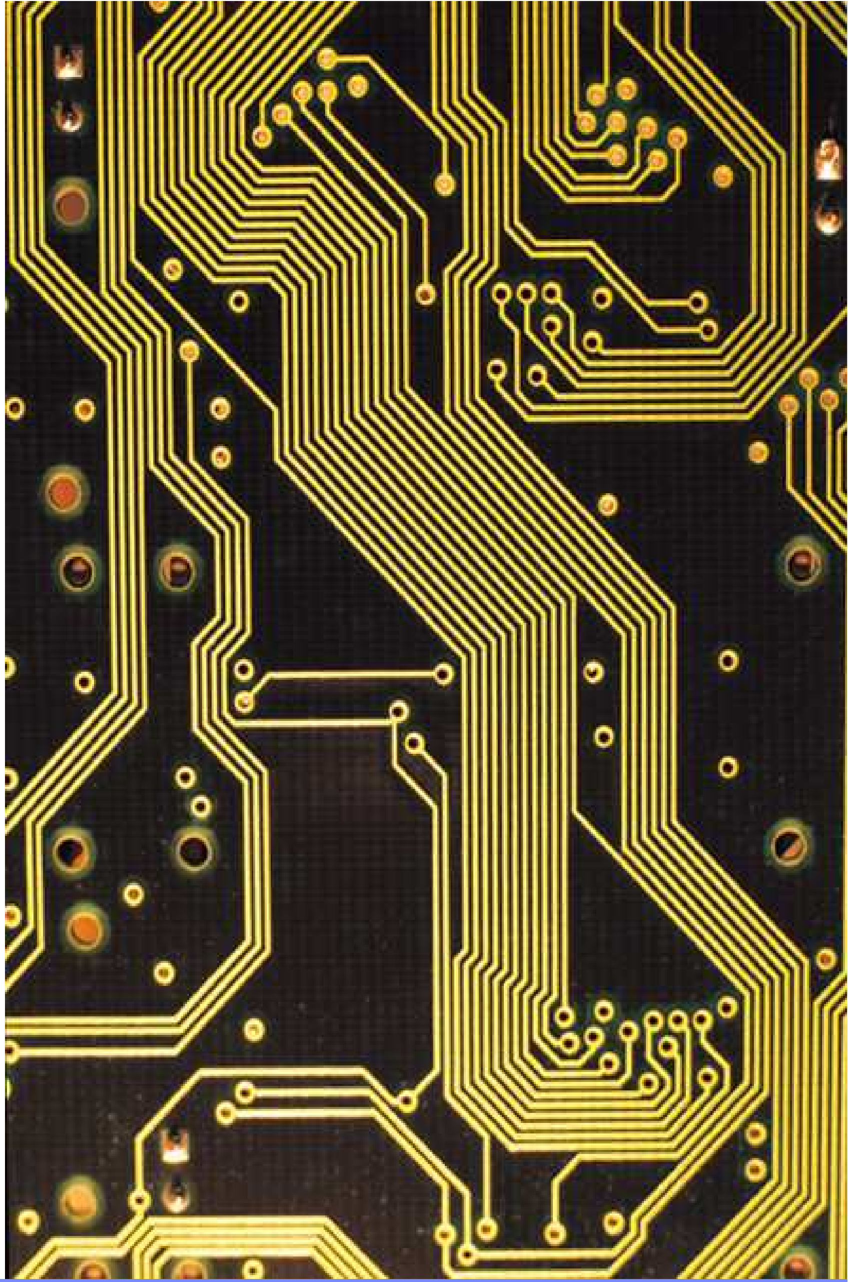


Compiler

- lexical analysis
  - scanner, symbols
- syntactical analysis
  - right sequence of symbols
- semantical analysis
  - correct types
  - names within the scope of application
  - static semantic
  - dynamic semantic → runtime environment
- code generation
  - object code
  - close to the architecture → machine code

Linker

- binder
- executable program
- relative addressing
  - computed to the beginning of a file
- linker map
  - add the address ranges from other files
  - common address range for the program
- executable program

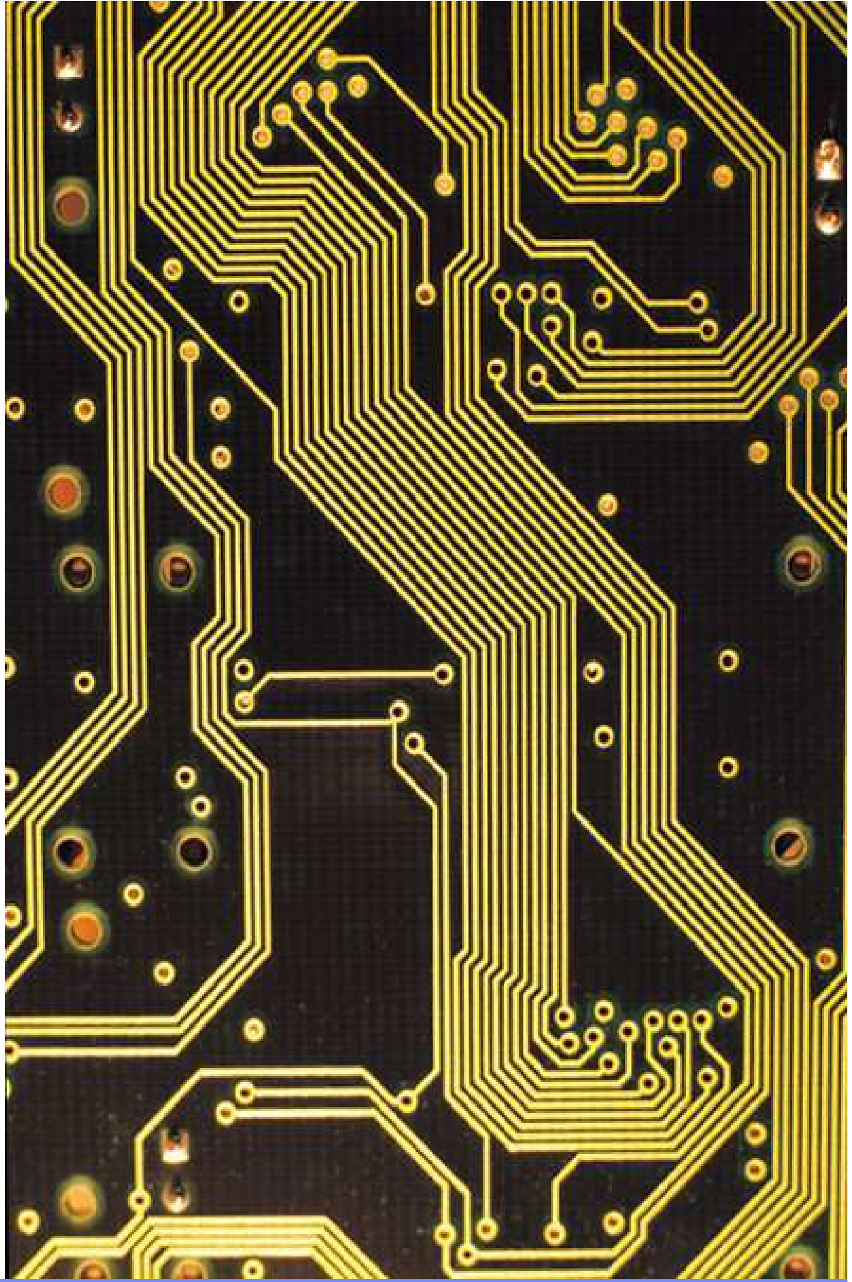


Runtime Environment

- all subroutines to execute the program
- interaction with the operating system
    - memory allocation
    - input / output
- memory management
    - stack, heap
- dynamic semantic
- error reporting
    - core dump
- threats, exceptions

Loader

- program goes to the computer's memory
- virtual addresses
- memory management



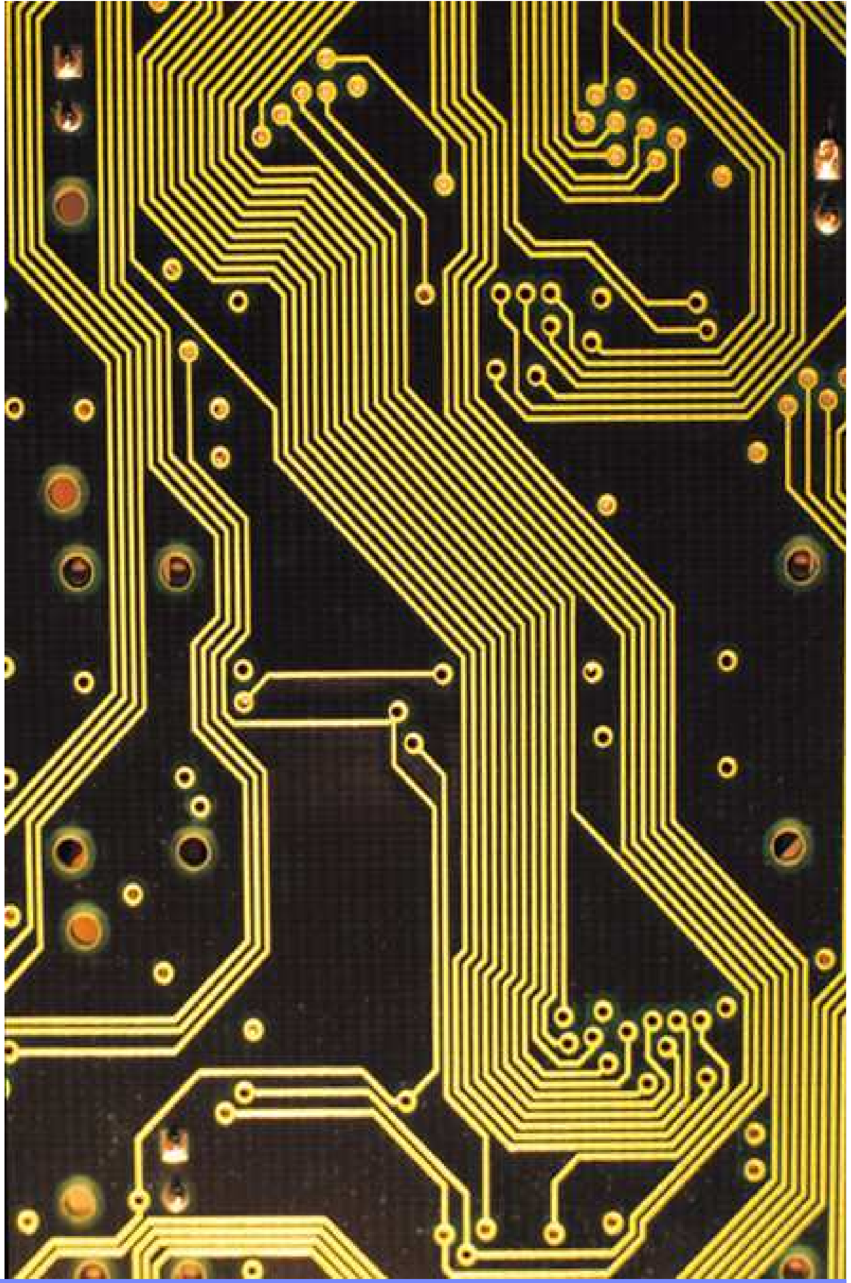
Debugger

- error analysis
- error detection
- stop points
- variable values
- no substitution for methodical programming

Programming is not coding!!!

Development Environment

- integrated environment for software development
- compiler, linker, loader, debugger, editor
- project management



Additional Information

- ACSII Code
  128 characters
- enhanced ASCII Code
  256 characters
- EBCDIC Code
  256 chcracters
- nationally adapted codes
  e.g. country specific enhanced
  ASCII Code

ASCII Code

0		**16**	►	**32**		**48**	0	**64**	@	**80**	P	**96**	`	**112**	p
1	☺	**17**	◄	**33**	!	**49**	1	**65**	A	**81**	Q	**97**	a	**113**	q
2	☻	**18**	↕	**34**	“	**50**	2	**66**	B	**82**	R	**98**	b	**114**	r
3	♥	**19**	‼	**35**	#	**51**	3	**67**	C	**83**	S	**99**	c	**115**	s
4	♦	**20**	¶	**36**	$	**52**	4	**68**	D	**84**	T	**100**	d	**116**	t
5	♣	**21**	§	**37**	%	**53**	5	**69**	E	**85**	U	**101**	e	**117**	u
6	♠	**22**	▬	**38**	&	**54**	6	**70**	F	**86**	V	**102**	f	**118**	v
7	•	**23**	↨	**39**	´	**55**	7	**71**	G	**87**	W	**103**	g	**119**	w
8	◘	**24**	↑	**40**	(	**56**	8	**72**	H	**88**	X	**104**	h	**120**	x
9	○	**25**	↓	**41**	)	**57**	9	**73**	I	**89**	Y	**105**	i	**121**	y
10	◙	**26**	→	**42**	*	**58**	:	**74**	J	**90**	Z	**106**	j	**122**	z
11	♂	**27**	←	**43**	+	**59**	;	**75**	K	**91**	[	**107**	k	**123**	{
12	♀	**28**	∟	**44**	,	**60**	<	**76**	L	**92**	\	**108**	l	**124**	|
13	♪	**29**	↔	**45**	-	**61**	=	**77**	M	**93**	]	**109**	m	**125**	}
14	♫	**30**	▲	**46**	.	**62**	>	**78**	N	**94**	^	**110**	n	**126**	~
15	☼	**31**	▼	**47**	/	**63**	?	**79**	O	**95**	_	**111**	o	**127**	⌂

ASCII Code with 128 characters

2nd Part of enhanced ASCII Code

128	Ç	**144**	É	**160**	á	**176**	░	**192**	└	**208**	ð	**224**	Ó	**240**	-
129	ü	**145**	æ	**161**	í	**177**	▒	**193**	┴	**209**	Ð	**225**	ß	**241**	±
130	é	**146**	Æ	**162**	ó	**178**	▓	**194**	┬	**210**	Ê	**226**	Ô	**242**	‗
131	â	**147**	ô	**163**	ú	**179**	│	**195**	├	**211**	Ë	**227**	Ò	**243**	¾
132	ä	**148**	ö	**164**	ñ	**180**	┤	**196**	─	**212**	È	**228**	õ	**244**	¶
133	à	**149**	ò	**165**	Ñ	**181**	Á	**197**	┼	**213**	ı	**229**	Õ	**245**	§
134	å	**150**	û	**166**	ª	**182**	Â	**198**	ã	**214**	Í	**230**	µ	**246**	÷
135	ç	**151**	ù	**167**	º	**183**	À	**199**	Ã	**215**	Î	**231**	þ	**247**	¸
136	ê	**152**	ÿ	**168**	¿	**184**	©	**200**	╚	**216**	Ï	**232**	Þ	**248**	°
137	ë	**153**	Ö	**169**	®	**185**	╣	**201**	╔	**217**	┘	**233**	Ú	**249**	¨
138	è	**154**	Ü	**170**	¬	**186**	║	**202**	╩	**218**	┌	**234**	Û	**250**	·
139	ï	**155**	ø	**171**	½	**187**	╗	**203**	╦	**219**	█	**235**	Ù	**251**	¹
140	î	**156**	£	**172**	¼	**188**	╝	**204**	╠	**220**	▄	**236**	ý	**252**	³
141	ì	**157**	Ø	**173**	¡	**189**	¢	**205**	═	**221**	¦	**237**	Ý	**253**	²
142	Ä	**158**	×	**174**	«	**190**	¥	**206**	╬	**222**	Ì	**238**	¯	**254**	■
143	Å	**159**	ƒ	**175**	»	**191**	┐	**207**	¤	**223**	▀	**239**	´	**255**	

enhanced part of ASCII Code

Table of contents

Variables in C

- a variable has four characteristics
  - name
  - data type
  - value
  - address
- numeric data types
  - char
  - short
  - int
  - float
  - double
- derived data types
  - struct
  - union
  - pointer
  - arrays

Details about data types and variables will follow later on.

Table of contents

Preferences of C

- C is applicable for common applications
- efficient was of putting algorithms in place
- sufficient number of control structures
- many data types
- powerful amount of operators
- operators are not applicable for complex objects (e.g. strings) exception: structures
- no instructions for input and output
- great portability (there may be other languages with higher portability)
- modular programming (modular compiling)
- program code is very close to the architecture

C – one of the most recommended programming languages for programming close to the architecture.

Preferences of C

- C has an imperative style
- Imperative programming languages are

– closest to the architecture	assembler
– procedural	FORTRAN, Pascal, C
– object oriented	Smalltalk, Java, C++

- Declarative languages
  The desired result is described directly and a translator engine has to generate the processing steps
  - LISP, Prolog

Preferences of C

- lexical conventions
  - predefined characters within C
  - lexical units
  - names
  - keywords (reserved)
  - literal and symbolic constants
  - control codes

One Character is different from another character object.

Preferences of C

- lexical units
  - tokens
  - parser
- lexical elements
  - names
  - reserved keywords
  - literal constants
  - constant character strings
  - operators
  - grammar elements

The parser analysis the program code and generates an interim code that consists of lexical elements.

Preferences of C

- C is a case sensitive programming language
  - `counter` and `Counter` are different
- dividing characters
  - characters between the visible characters (blanks, tab stops, form feed, comments, operators, …)
- comments
  - /* this is a valid comment */
  - // this is C++ style of comments
  - one or more comments within another comment is not allowed!

Comments are the most helpful thing in program code! Don’t forget to comment on your code!

Preferences of C

- names
  - internal: only valid within one file (names of macros at the preprocessor section are internal, too)
  - external: names of variables and functions that are valid for more than one file
  - 31 characters are important for internal
  - 6 characters can be used for external

Style guide: all names are written in non capital characters; only constants are written in capital letters.

Preferences of C

- reserved keywords
  - ISO standard: 32 keywords
    - auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while
- constants
  - literal
  - symbolic

Style guide: all names are written in non capital characters; only constants are written in capitals.

Table of contents

Constants

- In C there are two types constants:
  - literal constants
  - symbolic constants

There is a big difference between these two types of constants.

Use of Constants

- The syntax allows usage of constants or constant expressions.
- Anywhere where the syntax allows constants or constant expressions you can use literal or symbolic constants.
- There are differences between symbolic and literal constants.

Symbolic Constants

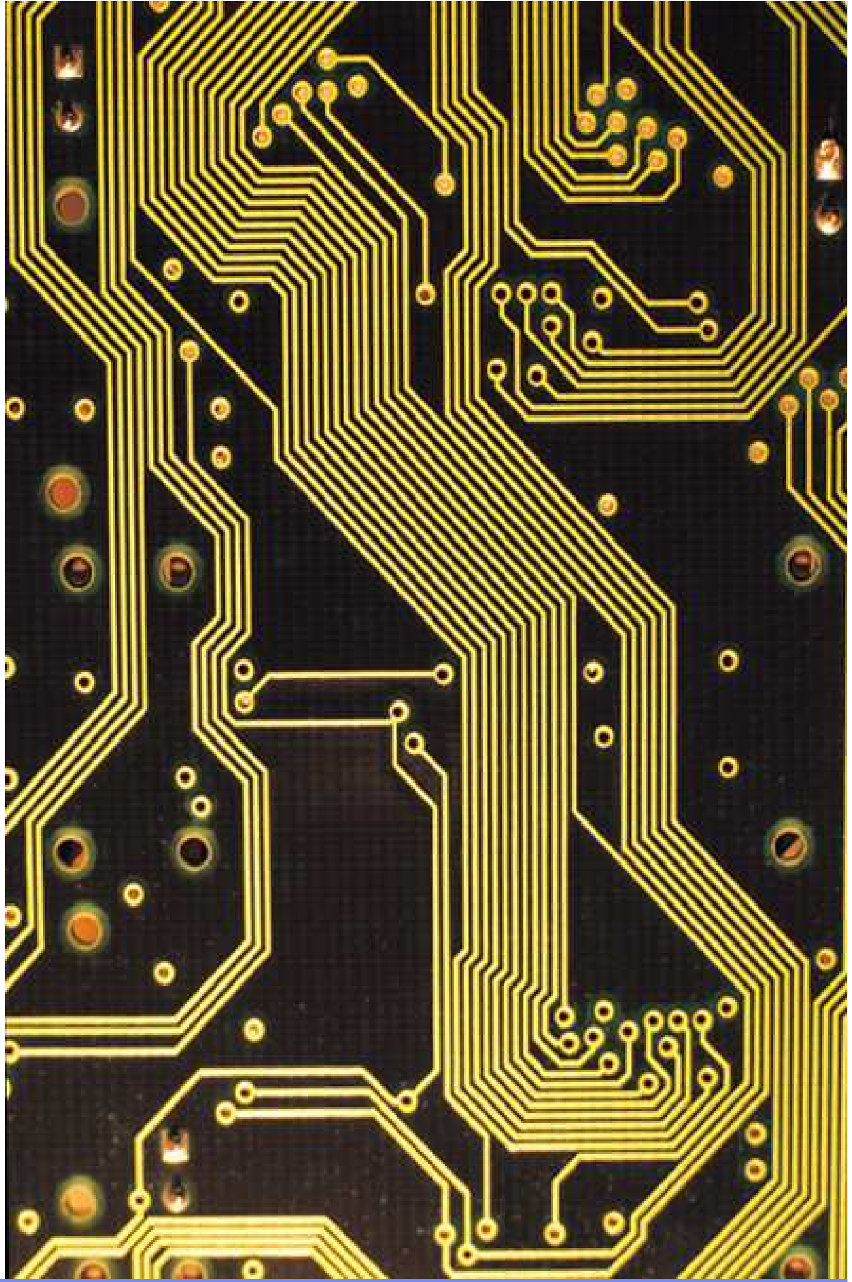
- Symbolic constants are defined by preprocessor keywords

```
#define PI 3.1415926
```

- This type of constant may be used for simplifying the change of constant parameters within your program code.
- If a parameter variable is used as a literal constant then you have to change each appearance of this parameter.

> *You have to compile the program again after changing the symbolic constant.*
>
> *Are you sure you haven't forgotten one?*

Literal Constants



- constants of integer type
- floating point constants
- enumeration constants
- constants of string type

Each of these constants have a data type, such as int, float, char, …

Literal Constants of Integer Types

- Integer constants like 1234 that are compatible with the data type `int` are of this data type.
- Integer constants can be displayed within different numbering systems, e.g. decimal, binary, octal, hexadecimal
- A number is octal if there is a `0` (zero) on the first position not followed by a ‘x’ or ‘X’ character.
  → `036637`
- A number is hexadecimal if there is a `0` (zero) on the first position followed by a ‘x’ or ‘X’ character.
  → `0x3AC3F`

Integer Constants

- You can enforce an integer constant to be of a certain data type.

`12445L` → is of data type `long`

`2234l` → is of data type `long`, too

`123ul` → is of data type `unsigned long`

- If the value is greater than the data type can accept the next appropriate data type is used implicitly.

> *int* → *long int* → *unsigned long int*
>
> *int* → *unsigned int* → *long int* → *unsigned long int*

Floating Point Constants

- Some examples for floating point constants are:

```
300.0               300 in float
1E3                 1000 in float
3.E2
.55E-3
```

- The first part of the scientific notation (before E) is Mantissa the second part (after the E) is the exponent.

> *Every floating point constant is of data type float by default. 10.0 is float, 10.0f is float, too.*

Enumeration Constants

- One example for enumeration constants is `TRUE` and `FALSE`.

```
enum boolean {FALSE, TRUE};
```

- The first value in enumerations has the numeric value 0 (zero) the next one 1, then 2, and so on …

```
TRUE +1 = FALSE
```

- The data type of the value `TRUE` (and `FALSE`, too) is `int`.

> *enum test {ALPHA, BETA, GAMMA};*
>
> *enum test {ALPHA=5, BETA=3, GAMMA=7};*

Constants of String Type

- Character constants have only one character in it, indicated by single quotation marks.

  `‘a’          ‘b’              ‘+’`

- Constants of string type mean more than one character, indicated by quotation marks

  `“hello world!\n”`

- Escape sequences are alternative representations

  `‘\n’` one character with CRLF meaning

Although the string constant is placed in memory as char type, the programmer accesses an int type.