

```

Line      = number statement "\\n";
statement = PRINT printlist |
           [LET] var "=" expression |
           GOTO expression |
           GOSUB (expression | characterstr) |
           RETURN |
           IF expression relop expression (THEN statement [ELSE statement][THEN] statement) |
           FOR var "=" expression ( TO | DOWNTO ) expression [STEP expression]|
           NEXT var |
           REM { any_character } |
           DIM var "(" expression ")" |
           SRAND |
           WAIT expression |
           OUT "(" characterstr "," expression ")" "=" expression |
           EPOKE "(" expression ")" "=" expression |
           VPOKE "(" characterstr ")" "=" expression |

```

# uBasic

*Ein plattformunabhängiger BASIC-Interpreter*

Uwe Berger; 2011

```

pr      =
pr      = (expression | characterstr)
va      = var | var "," varlist;
ex      = expression | expression "," exprlist;
ex      = ["-"] "+" (term | "(" term ")");
term    = factor | factor operator factor;
fact    = "(" expression ")" |
          ABS "(" expression ")" |
          NOT "(" expression ")" |
          EPEEK "(" expression ")" |
          ADC "(" expression ")" |
          IN "(" characterstr "," expression ")" |
          CALL "(" characterstr "," expression ")" |
          VPEEK "(" characterstr "," expression ")" |
          VPOKE "(" characterstr "," expression ")" |
          VPRINT "(" characterstr "," expression ")" |
          VPRINT "(" characterstr "," expression "," expression ")";
number  = decnumber | hexnumber | dualnumber;
decnumber = decdigit {decdigit};
hexnumber = "0" ("x"|"X") hexdigit {hexdigit};
dualnumber = "0" ("b"|"B") dualdigit {dualdigit};
separator = "," | ";" | ":";
var      = ("A" | "... " | "Z" | "a" | "... " | "z") ["(" expression ")"];
decdigit = "0" | "... " | "9";
hexdigit = "0" | "... " | "9" | "A" | "... " | "F" | "a" | "... " | "f";
dualdigit = "0" | "1";
relop    = "<" | ">" | "=" | "<=" | ">=" | "<>" | "<>";
operator = "+" | "-" | "*" | "/" | "%" | "mod" | "|" | "or" | "&" | "and" | "xor" | "shl" | "shr";
characterstr = "\" { any_character } "\"";

```



# Uwe Berger

- Beruf: Softwareentwickler (PPS-Systeme)
- Linux seit ca. 1995
- Freizeit: Hard- und Softwarespielereien
- Brandenburger Linux User Group e.V. (BraLUG)
- Brandenburger Linux Infotag (BLIT)  
→ 8.BLIT am 05.11.2011 in Potsdam



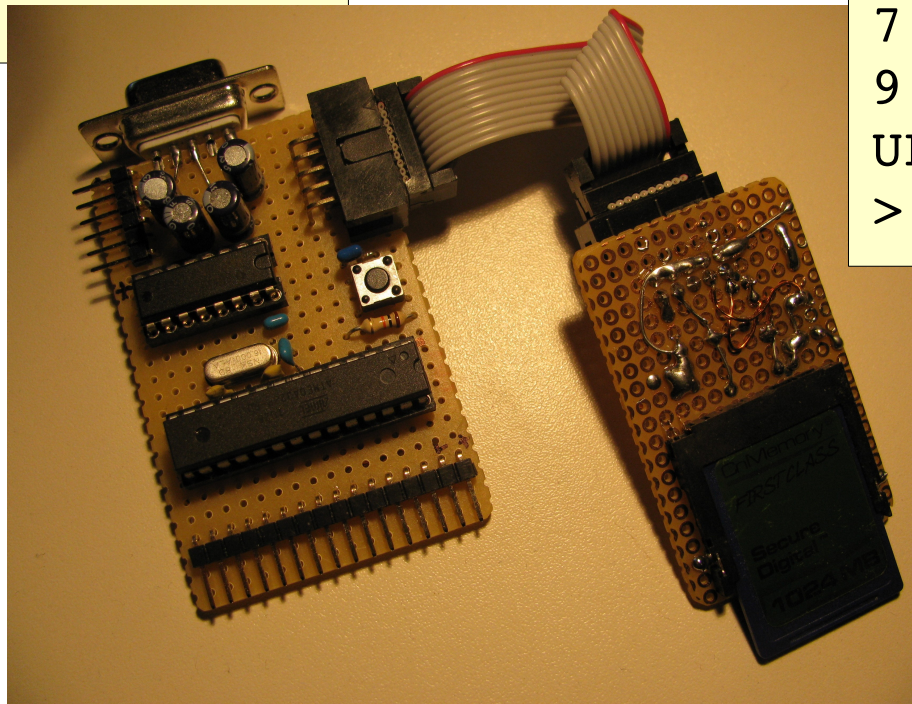
# Inhalt

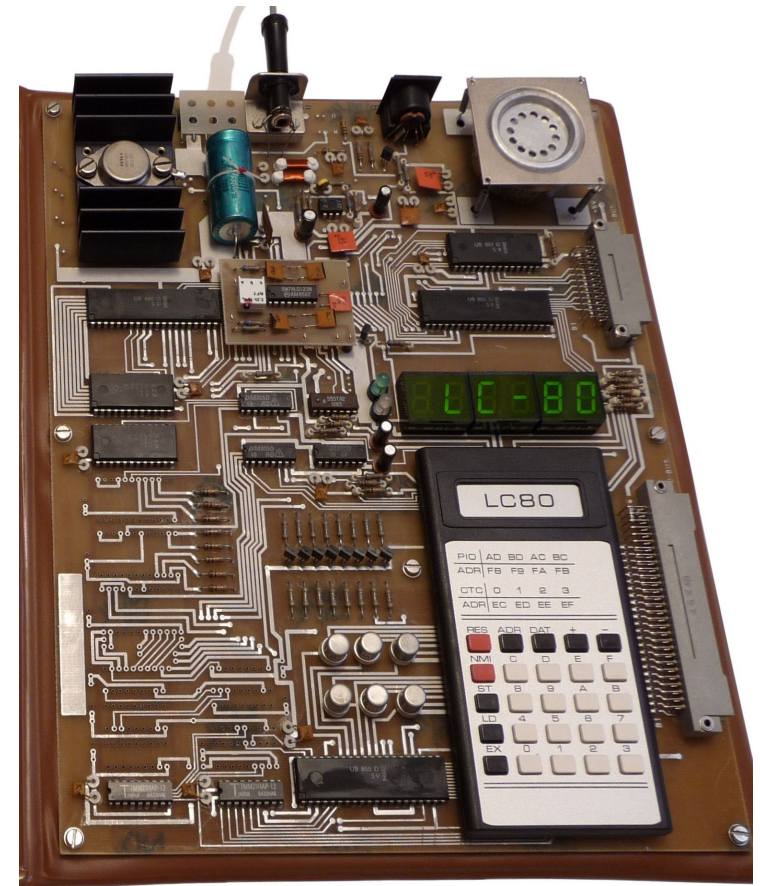
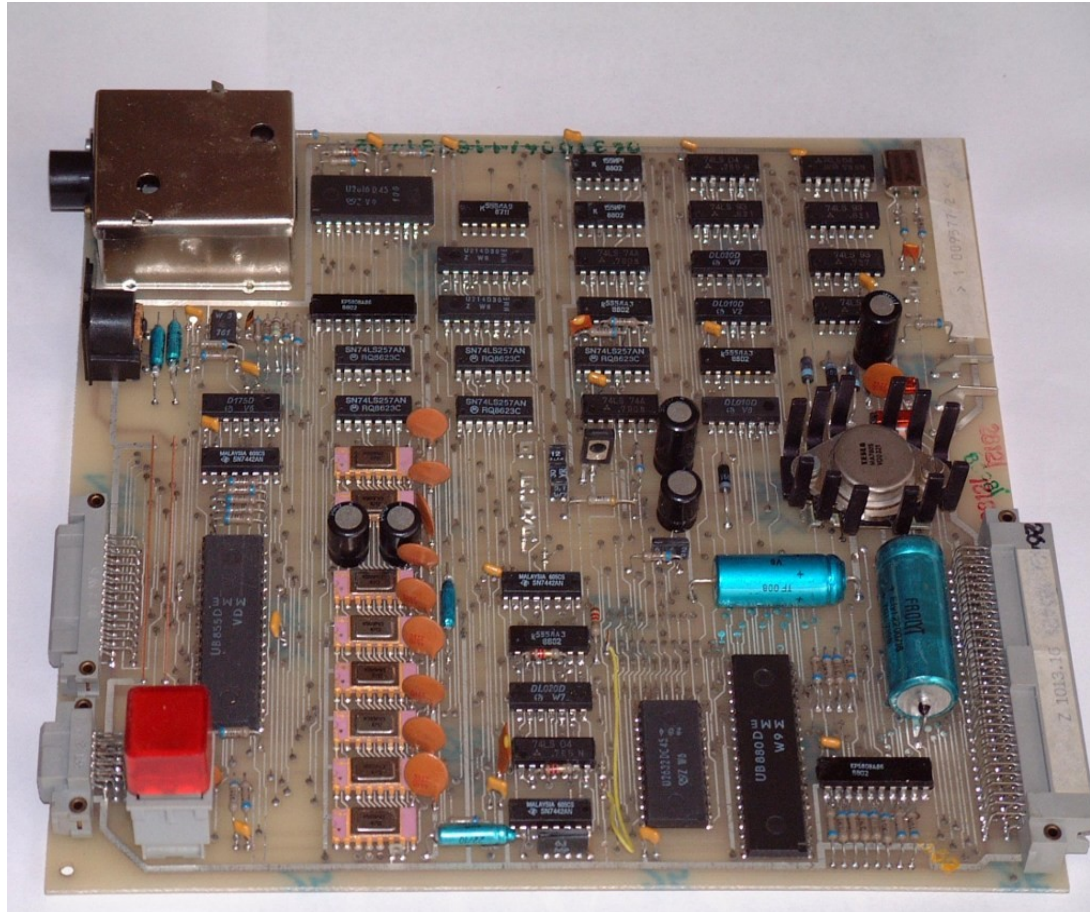
- Warum ein weiterer Interpreter?
- Etwas Theorie...
- uBasic



```
10 PRINT "Hallo CLT!"
20 FOR A=1 TO 10
30 IF A%2 THEN PRINT A
40 NEXT A
50 GOSUB 100
60 END
100 PRINT "UP"
110 RETURN
```

```
Hallo CLT!
1
3
5
7
9
UP
>
```





Bildquellen: Wikipedia





Warum?



# Was wurde gesucht?

- eine Möglichkeit bestehende Applikationen einfach um weitere Funktionalität erweitern zu können ohne sie neu übersetzen zu müssen
- ein Interpreter, der auch auf ressourcenarmen Plattformen läuft (z.B. AVR-MCU)
- ein Interpreter der leicht erweiterbar und konfigurierbar ist
- eine Codebasis für die unterschiedlichsten Hardwareplattformen
- ein allgemein bekannter Sprachsyntax → BASIC
- eine „akademische“ Herausforderung...



# Was gibt es schon für AVR-MCUs?

- AVR-ChipBasic: <http://www.jcwolfram.de>
- BasicBeetle: <http://www.dieprojektseite.de>
- ...?

Aber:

- es handelt sich um vollständige und abgeschlossene „BASIC-Computer“
- Firmware ist gänzlich in Assembler geschrieben
- Code ist nicht portierbar





## ...und was gibt es noch?

- gefunden wurde aber auch eine ideale Ausgangsbasis für eine Eigenentwicklung:

**Adam Dunkle: „uBASIC – A really tiny BASIC interpreter“**

(<http://www.sics.se/~adam/ubasic/>)

- auch neulich gefunden...: uBasic in CHDK (**C**anon **H**ack **D**evelopment **K**it) → <http://chdk.wikia.com/wiki/UBASIC>



# Theorie...

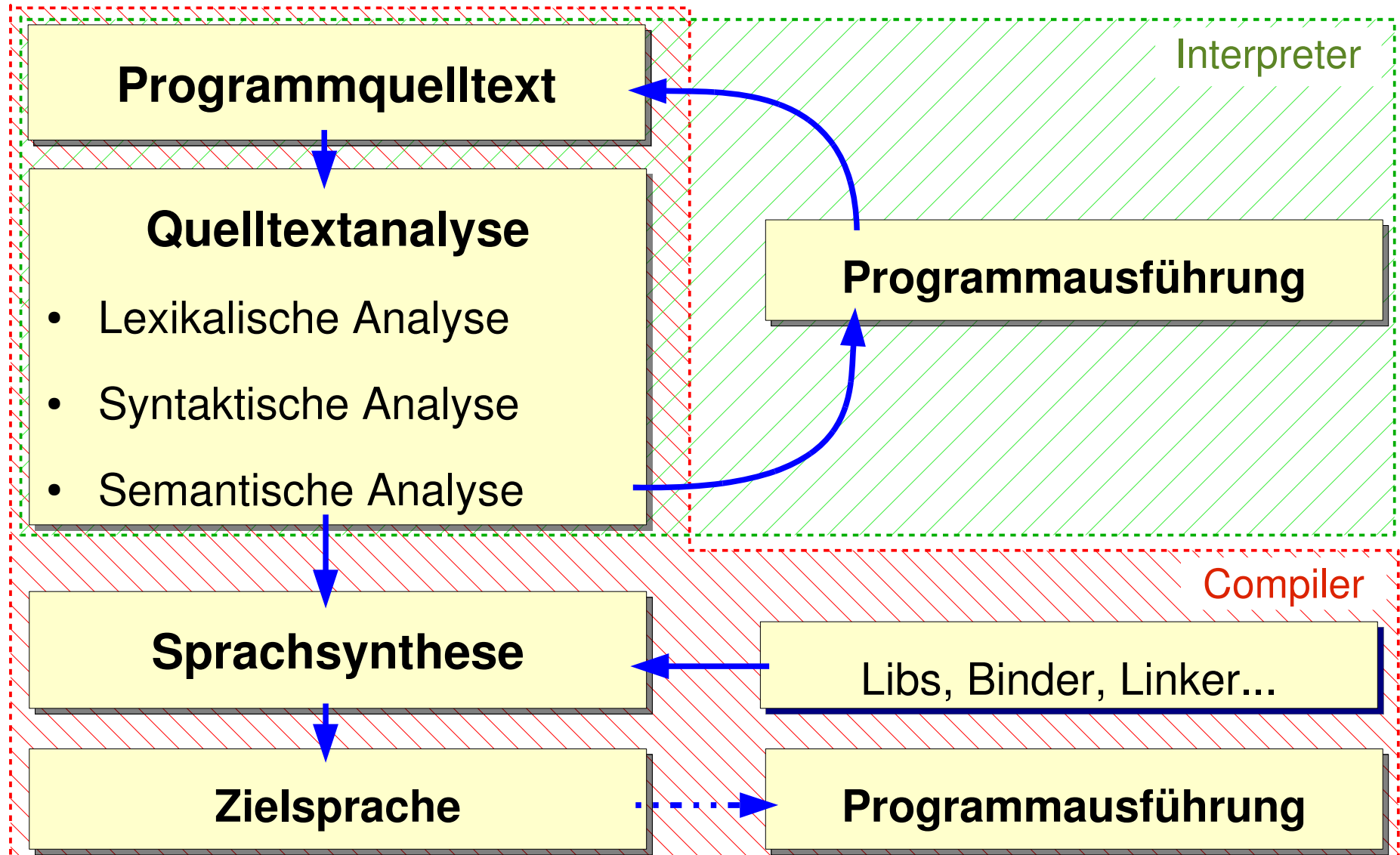


# Compiler, Interpreter

- Interpreter:
  - analysiert ein Quellprogramm unmittelbar zur Laufzeit und führt die einzelnen Befehle sofort aus
  - das Quellprogramm liegt in der Regel in einer lesbaren Form vor
- Compiler:
  - übersetzt ein Programm von einer Quellsprache in das semantische Äquivalent einer Zielsprache
  - in der Regel wird als Quellsprache eine lesbare Form und als Zielsprache meist direkt ausführbarer Code verwendet
  - die Ausführung des Programms erfolgt asynchron nach der Übersetzung



# Compiler, Interpreter (Funktionsweise)





# Vor-/Nachteile Interpreter

- Vorteile:
  - Quellprogramm plattformunabhängig, es wird lediglich ein plattformspezifischer Interpreter benötigt
  - effizientere Entwicklungsphase (auch Fehlersuche), da der Übersetzungsvorgang entfällt → das was im Quelltext steht, wird auch wirklich ausgeführt
- Nachteile:
  - geringere Ausführungsgeschwindigkeit, da Quelltext während der Ausführung (teilweise mehrmals → z.B. Schleifen) analysiert werden muss



# Vor-/Nachteile Compiler

- Vorteile:
  - hohe Ausführungsgeschwindigkeit, da Zielsprache in der Regel maschinennah
  - zeitaufwendige Quellcodeanalyse erfolgt einmal zum Zeitpunkt der Übersetzung
  - plattformspezifische Optimierungsmöglichkeiten
- Nachteile:
  - erzeugter Code ist nicht plattformunabhängig, für jede Plattform wird eine eigene Toolchain benötigt
  - zur Fehlersuche werden immer plattformspezifische Werkzeuge benötigt



# „Zwischen“ Compiler und Interpreter...

- JIT-Compiler:
  - Just-in-Time
  - Programmcode wird zur Laufzeit vollständig in Maschinencode übersetzt und danach sofort ausgeführt
- Bytecode-Interpreter:
  - Quellcode wird in einen einfacheren, optimierten und plattformunabhängigen Zwischencode übersetzt
  - die Ausführung des Bytecode erfolgt in einem plattform-spezifischen Interpreter (VM)





# Compiler-/Interpreterbau (Hilfsmittel)

- siehe diverse Standardliteratur der Informatik
- Syntaxbeschreibung → z.B. Backus-Naur-Form
- Werkzeuge zur Generierung von Quelltextparser:
  - Programmierung von Hand ;-)
  - automatische Parser-Generatoren: Lex, Flex, Yacc, Bison etc.
- Werkzeuge zur automatischen Compilergenerierung (Sprachsynthese) sind derzeit noch Forschungsgegenstand



# Die Sprache BASIC

- BASIC → Beginner's All-purpose Symbolic Instruction Code
- imperative Programmiersprache: eine Berechnung wird durch die Folge von Anweisungen beschrieben
- 1964: John George Kemeny, Thomas Eugene Kurtz
- Ansatz:
  - „*Programmieren lernen ohne Vorkenntnisse.*“
  - „*Die Hardwarespezifika sind nicht Gegenstand des Programms.*“
- sehr weite Verbreitung durch Heimcomputer der 80'er



uBasic...



# uBasic: Merkmale

- Interpreter ist vollständig in C geschrieben
- Schnittstellen/Abstraktionsschichten zur einbettenden Applikation vorhanden
- einfach in eigene Applikationen integrierbar
- modularer und verständlicher Aufbau; leicht erweiterbar
- extrem ressourcenschonend (z.B. ATmega168: 16kByte Flash, 1kByte SRAM)
- Basic-Sprachumfang mit TinyBASIC vergleichbar
- „intelligente“ Mechanismen zur Laufzeitverbesserung



# uBasic: Sprachumfang

- Variablen und eindimensionale Felder (DIM)
- Dezimal-, Dual-, Hexadezimal-Zahlenformat
- Arithmetische, Bit- und Vergleichs-Operatoren
- Schleifen (FOR-NEXT); auch DOWNTO und STEP
- bedingte Anweisung (IF-THEN-ELSE)
- Sprünge (GOTO)
- Unterprogramme (GOSUB)
- diverse weitere Funktionen und Anweisungen (Ausgabe, Kommentar, Zufallszahlen usw.)



```

Line      = number statement "\\n";
statement = PRINT printlist |
          [LET] var "=" expression |
          GOTO expression |
          GOSUB (expression | characterstr) |
          RETURN |
          IF expression relop expression (THEN statement [ELSE statement][THEN] statement) |
          FOR var "=" expression ( TO | DOWNTO ) expression [STEP expression]|
          NEXT var |
          REM { any_character } |
          DIM var "(" expression ")" |
          SRAND |
          WAIT expression |
          OUT "(" characterstr "," expression ")" "=" expression |
          EPOKE "(" expression ")" "=" expression |
          VPOKE "(" characterstr ")" "=" expression |
          DIR "(" characterstr "," expression ")" "=" expression |
          CALL "(" characterstr "," exprlist ");";
printlist = [printitem | printitem separator printlist];
printitem = (expression | characterstr);
varlist   = var | var "," varlist;
exprlist  = expression | expression "," exprlist;
expression = ["-"|"+"] (term | "(" term ")");
term      = factor | factor operator term;
factor    = var | ["+" | "-"] number | expression | function;
function  = RAND "(" expression ")" |
          ABS "(" expression ")" |
          NOT "(" expression ")" |
          EPEEK "(" expression ")" |
          ADC "(" expression ")" |
          IN "(" characterstr "," expression ")" |
          CALL "(" characterstr "," exprlist ")" |
          VPEEK "(" characterstr ")" ;
number    = decnumber | hexnumber | dualnumber;
decnumber = decdigit {decdigit};
hexnumber = "0" ("x"|"X") hexdigit {hexdigit};
dualnumber = "0" ("b"|"B") dualdigit {dualdigit};
separator = "," | ";" ;
var       = ("A" | "... " | "Z" | "a" | "... " | "z") ["(" expression ")"];
decdigit  = "0" | "... " | "9";
hexdigit  = "0" | "... " | "9" | "A" | "... " | "F" | "a" | "... " | "f";
dualdigit = "0" | "1";
relop     = "<" | ">" | "=" | "<=" | ">=" | "<>" | "<>";
operator  = "+" | "-" | "*" | "/" | "%" | "mod" | "|" | "or" | "&" | "and" | "xor" | "shl" | "shr";
characterstr = "\" { any_character } "\"";
    
```



# uBasic: Einschränkungen

- maximal 26 Variablen (signed integer); Variablennamen nur einzelne Buchstaben
- ein BASIC-Befehl pro Programmzeile
- kein (expliziter) Eingabebefehl (INPUT)
- (noch) keine DATA/READ/RESTORE-Anweisungen
- kein DO-WHILE oder REPEAT-UNTIL
- keine Strings
- endlicher (aber konfigurierbarer) Speicherplatz für diverse interne Caches und Stacks





# uBasic: Funktionsweise

- „Klassisches“ Konzept der Quelltextanalyse
- tokenizer.c (der Parser)
  - Überlesen von Whitespaces im BASIC-Programm
  - Token-Erkennung → *lexikalische Analyse*
- ubasic.c (der eigentliche Interpreter)
  - schrittweises Zusammensetzen der erkannten Token nach den jeweiligen Vorgaben → *syntaktische Analyse*
  - Überprüfung der Richtigkeit des Syntax und der Randbedingungen → *semantische Analyse*
  - Ausführung der erkannten Anweisung oder Teile davon, sobald sie einen Sinn ergeben



# uBasic: Funktionsweise

```
static void dim_statement(void) {  
    int var, dim;  
    accept(TOKENIZER_DIM);  
    var = tokenizer_variable_num();  
    accept(TOKENIZER_VARIABLE);  
    accept(TOKENIZER_LEFTPAREN);  
    dim=expr();  
    variables[var].dim=dim;  
    variables[var].adr=malloc(dim*sizeof(int));  
    if (variables[var].adr == NULL ) {  
        tokenizer_error_print(current_linenum, 42);  
        ubasic_break();  
    }  
    accept(TOKENIZER_RIGHTPAREN);  
    accept(TOKENIZER_CR);  
}
```

```
10 DIM A(N*10)
```

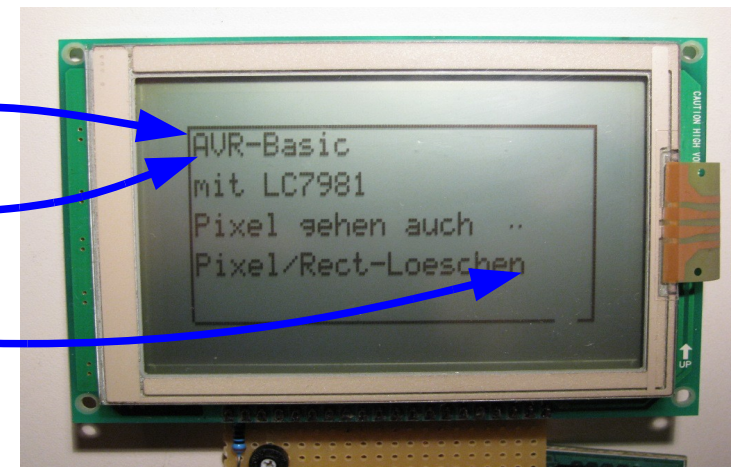
```
int expr(void) {  
    int val;  
    /* ... */  
    return val;  
}
```



# uBasic: externe Routinen/Variablen

- BASIC-Befehl zum Aufruf von Routinen der einbettenden Anwendung: CALL()
- BASIC-Befehl zum Lesen/Schreiben von Variablen der einbettenden Anwendung: VPEEK(), VPOKE()
- Routinen/Variablen müssen entsprechend dem BASIC-Interpreter bekannt gemacht werden

```
10 CALL("clear")
20 CALL("line", 10, 10, 149, 10)
30 CALL("puts", 12, 12, "AVR-Basic")
40 CALL("pset", 120, 40)
50 REM ...
60 END
```





# uBasic: Programmmedium

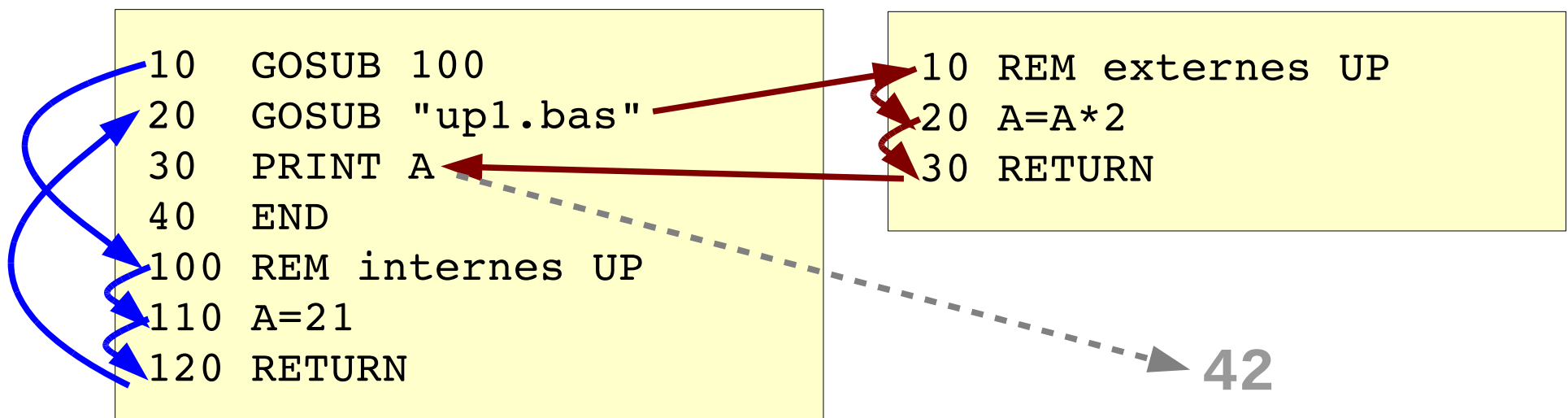
- der BASIC-Quelltext ist während der Abarbeitung ständig lesend im Zugriff (Parser/Tokenizer)
- der Quelltext kann/soll auf den unterschiedlichsten Medien stehen (RAM, SD-Karte, Festplatte, PROM etc.)
- die Zugriffe auf den BASIC-Quelltext lassen sich auf einige wenige Dinge reduzieren

```
#define PTR_TYPE          char const *
static PTR_TYPE          ptr;
#define PROG_PTR          ptr
#define SET_PROG_PTR_ABSOLUT(param) (PROG_PTR = (param))
#define GET_CONTENT_PROG_PTR *ptr
#define INCR_PROG_PTR    ++ptr
#define END_OF_PROG_TEXT GET_CONTENT_PROG_PTR == 0
```



# uBasic: GOSUB

- Standard: GOSUB n → Aufruf eines Unterprogramms ab der Zeile n; Rücksprung bei Auftreten eines RETURN
- Erweiterung: das Unterprogramm kann auch in einem externen „Quelltext-Bereich“ (z.B. Datei) liegen

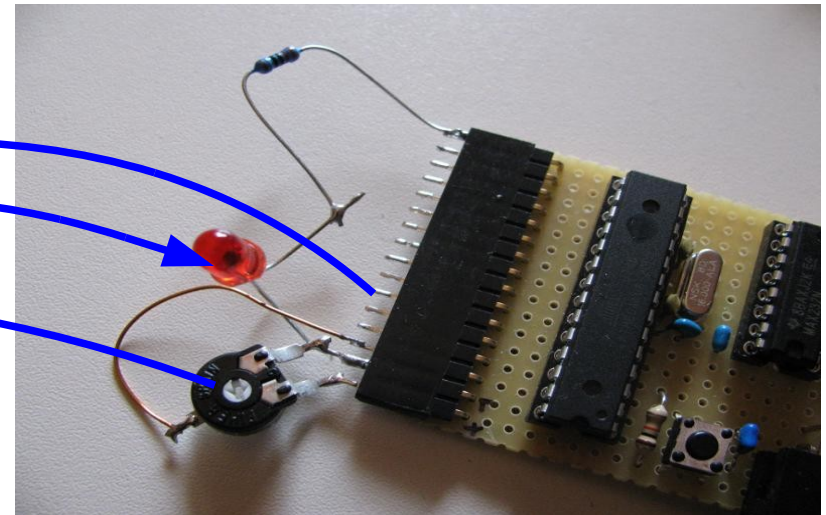




# uBasic: AVR-MCU-spezifisches

- Ein-/Auslesen von I/O-Ports: DIR() , IN() , OUT()
- Ein-/Auslesen des EEPROM: EPOKE(), EPEEK()
- BASIC-Befehl zum Auslesen des ADC: ADC()

```
10 DIR("b",1)=1
20 DIR("b",2)=0
20 A=0
40 PRINT IN("b",2)
30 OUT("b",1)=A
40 PRINT ADC(0)
50 IF A=1 THEN A=0 ELSE A=1
60 WAIT 1000
70 GOTO 30
80 END
```





# uBasic: Einbindung in Applikationen

- Konfiguration an Gegebenheiten anpassen
- entsprechende Header-Dateien inkludieren
- ... und ein paar Zeilen im Rahmenprogramm:

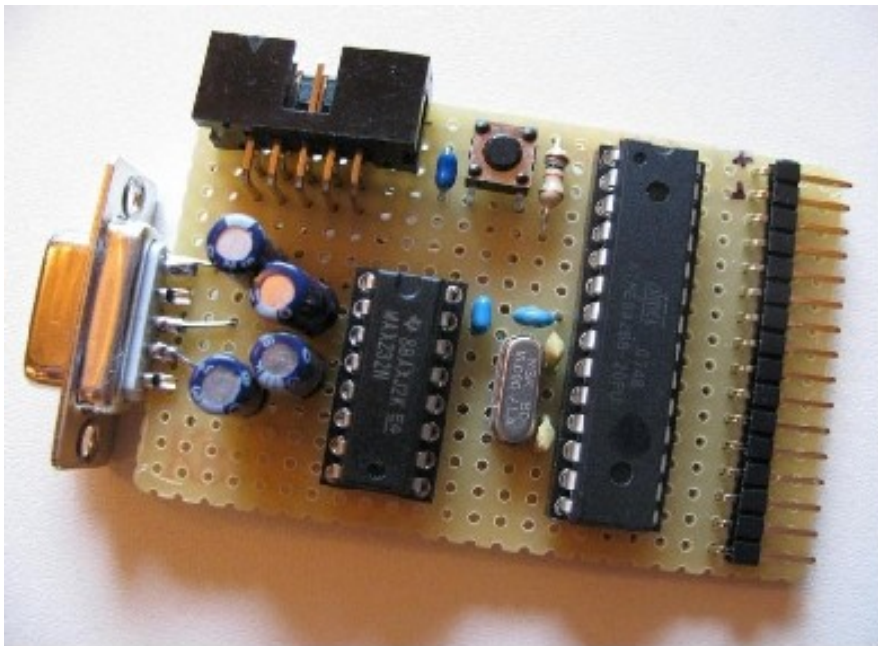
```
/* BASIC-Programm oeffnen */  
/* ... */  
  
ubasic_init(zeiger_auf_programmanfang);  
do {  
    ubasic_run();  
    /* ... was auch immer ... */  
} while(!ubasic_finished());
```





# uBasic: Live-Vorführung

- innerhalb einer Linux-Applikation
- als Firmwarebestandteil auf einem AVR-Mikrocontroller



```
~/work/ubasic_avr
Datei Bearbeiten Ansicht Terminal Beiter Hilfe
~/work/ubasic_avr
~/work/ubasic_avr
AVR-uBasic-Interpreter; Uwe Berger, 2010
Compiliert am May 7 2010 um 17:19:35
Compiliert mit GCC Version 4.3.0
list
10 gosub 100
15 a=30
20 for i = 1 to a step 10
30 print "i=", i
40 next i
50 end
100 print "subroutine"
110 return
run
subroutine
i= 1
i= 11
i= 21
```



# uBasic: zwei Versionen

- Uwe Berger
  - ressourcenoptimiert für Mikrocontroller
  - siehe dieser Vortrag...
- Rene Böllhoff
  - extrem hohe Modifizierbarkeit des Basic-Sprachsyntax via Defines/Makros
  - optionaler Pre-Compiler → Bytecode-Interpreter
  - dadurch teilweise bessere Performance, aber höherer Ressourcenverbrauch



# uBasic: Ausblick

- Bugfixes
- DATA/READ/RESTORE
- BASIC-Zeilen ohne Zeilennummern
- gebrochene Zahlen
- vielleicht doch ein „BASIC-Computer“ auf MCU-Basis, es fehlen doch nur noch Ein- und Ausgabegeräte...



## Weitere Informationen:

- <http://www.sics.se/~adam/ubasic/>
- <http://www.mikrocontroller.net/topic/177030>
- [http://www.mikrocontroller.net/articles/AVR\\_BASIC](http://www.mikrocontroller.net/articles/AVR_BASIC)
- <http://www.mikrocontroller.net/svnbrowser/avr-basic/>
- „Visionäre der Programmierung“; Biancuzzi & Warden; O'Reilly
- <http://www.ittybittycomputers.com/IttyBitty/TinyBasic/TBUserMan.htm>
- <http://www.ittybittycomputers.com/IttyBitty/TinyBasic/TBEK.txt>



Danke...!