TinyBASIC: Interactive programming system

Interpreter and step-by-step guide

Tomas Petricek, Charles University

tomas@tomasp.net
@tomasp.net
https://tomasp.net
https://d3s.mff.cuni.cz/teaching/nprg077

```
type Value = (* .. *)
type Expression = (* .. *)

type Command =
  (* Jumps and subroutines *)
  | Goto of int
  | GoSub of int
  | Return
  (* I/O operations *)
  | Clear
  | Print of Expression list
  | Input of string
  (* If, variables and control *)
  | If of Expression * Command
  | Assign of string * Expression
  | Run
  | Stop
```

BASIC interpreter structure (1/2)

Expressions evaluate to Values and are simple

Commands contain all the operations that modify the program state

Charles University

BASIC interpreter structure (2/2)

```
(* State of the interpreter stores
   program lines as sorted list,
   variables in a dictionary,
   generator for the RND function
   and stack for GOSUB/RETURN *)
type State =
  { Program : list<int * Command>
    Variables : Map<string, Value>
    Random : System.Random
    ReturnStack : int list }

(* Evaluate a command and then
   run the next one (if any)
   until the program ends.
 : State -> (int * Cmd) -> State *)
let rec runCommand state (line, cmd) =
  (* ... *)

(* Find the next line after 'line'
   and run that or stop if none *)
and runNextLine state line =
  (* .. *)
```

State is the program source code, variables (and a few extras)

Current line is also a part of the state (function argument)

Charles University

Demo

BASIC Hello World

```
REM You can write comments!
REM Jumping and calls
GOTO 10
GOSUB 10
RETURN

REM Printing to the screen
POKE 1024 CHR$(42)
PRINT "HELLO ";X
INPUT "ENTER A VALUE";X

REM Variables and ifs
X=10
IF (X>0) GOTO 10

REM Control
RUN
STOP
```

BASIC basics

GOSUB jumps, but keeps return location on stack for **RETURN**

PRINT takes a sequence of expressions (and we ignore cursor moving)

POKE writes a byte to memory (we will cheat)

We ignore command chaining (**:**)

Charles University

Demo

Elegant programs with GOSUB :-)

Lab overview

TinyBASIC system step-by-step

TinyBASIC - Basic tasks

1. Add `GOTO` and better `PRINT` for infinite loop fun!
   Evaluation of expressions, finding of the next line
2. Implement interactive program editing
   Handle commands that edit the program code
3. Add variables, conditionals, integers and bools
   Needs `Map<string, Value>` in the program state
4. Random function and (not quite correct) `POKE`
   To be able to generate random stars!
5. A few more functions and operators
   As required by the Nim (subtraction) game

Charles University

TinyBASIC - Bonus tasks

1. Add support for more elegant programs with GOSUB
   Needs `list<int>` (stack of return line numbers) in state
2. Refactor our Nim code sample to use it
   Dijkstra will still not be happy, but we avoid repetition
3. Implement an "AI" player for our Nim game
   Wikipedia says this is a solved problem :-)

Lessons learned

Interactive programming systems

- ≠ Evaluation logic not that far from TinyML!
- Imperative interpreter needs much more state
- How exactly interactive editing worked?!
- </> Parsing & interactive editing out of scope :-(

Charles University