Practical Exam Task

Prof. Dr. Peter Heusch

Introduction

Implement a Compiler that compiles selected features of GW-BASIC into CPL (C-Processor-Language). The features are defined in section 1.

Your compiler shall compile a complete GW-BASIC file into a single file consisting of two parts:

- The first part contains the assembler code of the CPL processor. The first part is the only part of the program that may depend on the GW-BASIC code. The format of this part is also defined in section 2.
- The second part contains supporting routines that may be written in C. This part must not depend on GW-BASIC, irrespective of the length of the GW-BASIC program.

1. Features that must be translated from GW-BASIC

Variables:

A variable identifier starts with a name which contains from 1 to 8 characters. The next part of a variable identifier is a type designator, which is one of the following characters:

- The character $ denotes a string variable
- The character % denotes an integer variable
- The character # denotes a double variable

The last part of a variable identifier is optional and denotes an integer expression in () used for adressing a specific element inside an array. Furthermore it can be used to adress a specific character inside a string. In this case it is possible to have double () to adress a character inside an element of a string array.

It is not possible for a variable identifier to denote an array as well as a scaler variable.

A variable is declared by simply assigning it a value, except for array variables, these must be declared by `DIM <Name>(<Elements>)` , where `<Elements>` is an integer literal expression. Redefinition of an array is not supported. Any array access must be range checked, i.e. the runtime environment must issue an error message if someone tries to address element 11 in a 10 element array.

Using the value of an undeclared variable (irrespective of whether it is a scalar value or an array value) or assigment to an undeclared array element shall be detected. It is up to you whether you check this at compile- or at runtime and whether the definition must precede the first usage also inside the source code or only at runtime. The validity of an array index resp. string index shall be detected at runtime.

Assignments

An assigment consists of of a variable identifier on the left side of the equals sign and an expression on the right side. As the CPL processor cannot automatically translate beteween different data types the compiler for the GW-BASIC program must explitly call the appropriate conversion function.

Expressions

An expression can be constructed from the following elements:

- At the lowest level there are
  - Variable identifiers
  - Integer, double and string literals
  - Nullary functions, denoted by an empty () after the function name
- At the next level there are unary and binary functions, whose arguments must be put in () and separated by , (for binary functions)
  - SIN, COS, TAN: the trignometric functions
  - LOG, EXP, SQR: the logarithmic, exponential and square root functions
  - CONCAT: function to concatenate two strings
  - LENGTH: function that returns the length of a string or the length of an array
  - DSTR, ISTR: functions to convert from integer and double to string
  - DINT, SINT: functions to convert from double and string to int
  - IDBL, SDBL: functions to convert from integer and string to double
  - INPUT, PRINT: functions to read or write text from stdin or to stdout
  - REM: comments
- Last not least expressions can be combined by arithmetic operators (in decreasing priority):
  - (and) for parenthesized operators
  - Unary +,- (positive or negative sign)

    - Binary ** (power)
    - Binary *,/,% (muliplication, division, remainder)
    - Binary +,- (addition, subtraction)
    - Binary comparison <, <=, <>, =, >=, >

Strings can be compared using the comparison operators or modified using the

- operator, all other operators are only defined for integer or double values.

Control structures

- You have to translate the following control structures:
    - The goto statement with a line number as a target.
    - The if statement with or without else, each controlling either a single assigment, a function call or a goto statement. The condition of an if statement must be a comparison between two values (either a variable identifier or a literal of the same type)
    - The for loop with arbitrary integer start, finish and (optional) step values.
    - The while loop, whose condition must be a comparison between two values (either a variable identifier or a literal of the same type)
    - The gosub return statement with a line number as a target.
    - The stop statement to halt the execution of the program

Implementation of the compiler

Your compiler must output PCL (Primitive C Language). In this language a program contains of a global data section and a single main-function in C.

The global data section

- Eight arrays, for each you can define the number of elements:
    - A global integer array named ints (for scalars) and inta (for arrays)
    - A global double array named reals (for scalars) and reala (for arrays)
    - A global string array named strings (for scalars) and stringa (for arrays)
    - A global void pointer array named voids (for scalars) and voida (for arrays)
- Four integer variables a, b, c, d usable to address array elements

The main function

- Three control structures:
    - Labels, as defined for labeled while and labeled for in Java

    - Goto statements redirecting the execution flow to the designated label, either by directly mentioning the label or by indirectly jumping to a value stored in one of the elements of voids.
    - Return statements with the return value 0.
- Assignments that either assign a literal or a single global variable to a global variable, e.g. `ints[5]=25` or `ints[5]=ints[7]`
- Assignments that negate the value of a global variable, e.g. `ints[5]=-ints[5]`
- Assignments that apply an binary operation on two global variables assigning the result to a third global variable, e.g. `ints[6]=ints[8]*ints[10]`
- Assignments that assign the value of a label to one of the values in voids, e.g. `voids[7]=label_nexti`
- In every assignment it is allowed to replace the literal index with one of the variables a, ..., d, e.g. `reals[a]=reals[b]*reals[10]`
- Function calls, where you call a function with arbitrary many integer arguments, whose interpretation is up to the function.

Runtime environment

You may write any number of functions in C providing the runtime environment. These functions can have any number of parameters, but the return type must be `void`. The interpretation of the parameters is entirely up to you. You will almost certainly implementations for all BASIC functions, but also for things like range checking, etc. You are also allowed to create data structures that support your functions, e.g. to maintain a stack for nested loops, gosub/return, etc.

Moreover this runtime environment must be fixed, you are not allowed to add functions to the runtime environment that are created either directly or indirectly from the GW-BASIC program. However, you are allowed to create functions that support your program, e.g. for checking whether you are actually inside a gosub at the moment when you see a return statement.

Your compiler and runtime environment must be written in such a way that they never crash. They can stop the program compilation or execution if the GW-BASIC program is either syntactically or semantically wrong, but they must issue an error message before deliberately stopping the compilation or execution.

During the presentation on June 18th you shall prove that you are able to compile the aforementioned GW-BASIC language into PCL, show us how you handle the non-definitely specified behaviour. This includes e.g. what happens when constructs are not properly nested.

Solution requirements

In order to pass the exam with at least 4,0 your compiler must at least be able to recognize all token and parse all possible code expressions. The next step should be to recognize the context sensitive syntax:

- Every goto and gosub redirects to an existing line
- Every for is followed by a next, any while is followed by a wend
- Every return is preceeded by a line that is the target of a gosub
- Every variable that is used is also declared (scalar and array) The last two steps are the implementation of the full language:
- Assignment statements (including functions), goto and if
- Loop statements and gosub/return

Moreover I invite you to an ask-me-anything session on May 28th at 8 am via zoom.