zekraut.md

Ze kraut Programming Language

What is kraut?

kraut is a very small programming language that serves as a quick and efficient way to write small but effeective programms that are automatically compiled into x86 assembly. It has been inspired of the COBOL programming language but isn't nearly as complex as it which makes it way easier to use. You could call kraut a german prototype of a more modern COBOL.

kraut's compiler only supports 32bit NASM assembler thus it can be run on 32bit machines and on 64bit machines. Moreover, it only runs on Linux since the syscalls it has to make at the assembly level are only compatible with Linux.

How the compiler works

Every integer declaration is pushed on the stack and is getting referenced by the initial stack pointer (represented by the ebp) minus its offset which is stored at the c++ level in an integer map.

The compiler aka kraut.cpp has two ways to store string values. The first one can be at the compiler level which means that strings are stored in a stringmap and are output into the actual assembler file (you can see the NASM output in the *.s file) when they are referenced.

The other way is to reference the strings directly in the *.s file. This has the advantage that one is able to reference the actual assembler strings at a lower level than only relying on the string storage in the compiler.

The command "Drucke" one the one hand takes full advantage of the string concatenation at the c++ level to make it easier to concatenate strings in order to print them to the terminal.

"Gebe aus" on the other hand only references at an assembler level which makes it possible to work with strings that are not stored in the stringmap which the compiler uses for the most part.

How to compile

Compilation is quite easy. Just take advantage of the runnable shell script called krautscript and specify your krautfile and your programmname like that:

```
./krautscript hallowelt.krt hallowelt
```

Quite easy isn't it?

Krautprogrammstruktur

Kraut is divided into a declaration section (you could call it Deklarierungssektion since kraut is completely German) and the actual programming section (Programmsektion).

You dont have to specify the Deklarierungssektion since the compiler doesn't make it mandatory but in case you want to structure your programm in a better way, just write Deklarierungssektion at the beginning of the file.

The only command that one is able to give in this section is the define command (Definiere). Therefore, you can only define and declare strings and integers in this section.

Then there is the Programmsektion which one has to specify in a line beforehand.

This is the section where you will write your actual kraut program in.

The syntax

I made the syntax very strict in order to simplify the compiler and to concentrate more on the development of the actual language.

You have to watch out for these core parts of kraut's syntax:

- every command has to end with a dot like a real sentence
- every line only includes one command
- every program has to have a Programmsektion or the compiler will write nonsense to the NASM file
- watch out for unnecessary spaces! They will break your program!

hallo Welt!

To make this manual way easier to read I want to include an example of how to write a hello world program. It's quite easy.

```
Programmsektion
Drucke "hallo Welt!".
```

Working with integers

```
Definiere x=23.
Definiere hallo="einszweidrei".
Programmsektion
;This comment will be unnoticed by the compiler
x=x+2*3.
;x will be added to 2 (25) and then multiplied by 3 (kraut treats every operation like in brackets)
hallo[0]='n'.
hallo[1]='e'.
hallo[2]='u'.
hallo[3]='n'.
;the first four bytes where hallo points to are now "neun"
Gebe aus x.
; The command "Gebe aus" can output numbers at the assembler level which means you could also give out a register directly
;the compiler will translate x immediately to [ebp-4] which stores the value of x
Drucke hallo,"zehn".
; concatenates hallo and "zehn" to "einszweidreizehn" and prints it to the terminal
Setze Fehlernummer x.
```

```
;will output the value of x as the return code of the program. One could also just write a literal number instead of x or a register

; here is a little trick to modify strings at the assembler level while expanding it
hallo[12]=0x0a.
; hallo now includes a newline character at the index 12 which wouldnt be output by "Gebe aus" since NASM wouldnt know that the string size of hallo has increased
Gebe aus hallo(13).
; This will output hallo with the appended newline character since the number in the brackets specifies the length of the string that shall be output
```

If statements

All if statements (Wenn) have to end with "Mach weiter." (it's the same as end-if in this context). Everything that is inbetween the Wenn and Mach weiter will be executed in case the Wenn-statement is true.

```
Definiere x=23.
Programmsektion
Wenn x kleiner 24.
Drucke "hallo".
; Drucke wird ausgeführt, weil 23 kleiner 24 ist
Mach weiter.
```

I want to list all of the possible Wenn statements and their translations:

- Wenn x kleiner r : x < r
- Wenn x größer r or Wenn x groesser r : x > r
- Wenn x gleich r: x = r
- Wenn x ungleich r: x != r
- Wenn x größer/gleich r or Wenn x groesser/gleich r : x >= r
- Wenn x kleiner/gleich r : x <= r

While statements

Its syntax is similar to Wenn. Everything that is included inbetween Während and Mach weiter will get executed in case Während is true.

Keep in Mind that the compiler can see the difference between a Mach weiter of a Wenn statement and a Mach weiter of a Während statements which make it possible to combine the conditional statements with the while loop.

```
Definiere x=1.
Programmsektion
Während x kleiner 5.
Drucke "hallo".
x=x+1.
Mach weiter.
```

One can substitute Während with Waehrend to make it possible to write kraut programs on a non German keyboard.

Goto statements

Every line that ends with : will be treated as a label by the compiler which can be accessed with Spring in (the jump/goto statement).

```
Programmsektion
Springe in qwertz.
Drucke "1234".
qwertz:
Drucke "5678".
;1234 wont be printed since the jump statement jumps to qwertz
```

Embedding assembly into the code

In case you want to program at the assembly level to make it possible to implement functions in your program that kraut doesnt support yet, you can take advantage of the subsection of the Programmsektion by including "Assemblerstart" in a line and ending this subsection with "Assemblerende".

Everything that is inbetween those keywords will immediately be copied into the NASM file without any modification by the compiler.

Since they are just marking a subsection they shouldnt be ended with a dot!

```
Programmsektion
Drucke "test".
Assemblerstart
mov eax, 1
mov ebx, 87
int 0x80
Assemblerende
```

Writing to file

You can even write strings to a file by using "Schreibe".

```
Programmsektion
Schreibe "hallo Welt!" in "testdatei".
```

This will write hallo Welt! in the newly created file called testdatei.