

PicoC-Compiler

Übersetzung einer Untermenge von C in den Befehlssatz der RETI-CPU

Kolloquiumspräsentation

Präsentator:
Jürgen Mattheis

Gutachter:
Prof. Dr. Scholl

Betreuung:
M.Sc. Seufert

27. September 2022

Universität Freiburg, Lehrstuhl für Betriebssysteme

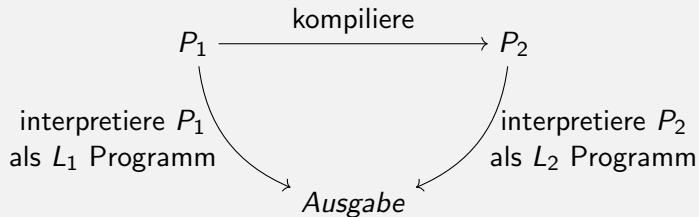
Einführung

Motivation

Compiler



- übersetzt ein Programm von einer Sprache L_1 in eine andere Sprache L_2 .
- beide Programme gleiche Semantik.



Aufgabenstellung

- ▶ L_C Programm kompilieren: `> gcc program.c -o machine_code .`

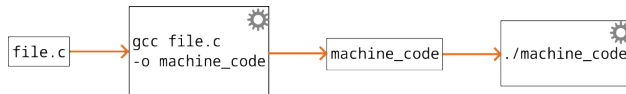


Abbildung 1: Schritte zum Ausführen eines Programmes mit dem GCC.

- ▶ L_{PicoC} Programm kompilieren: `> picoc_compiler program.picoc .`



Abbildung 2: Schritte zum Ausführen eines Programmes mit dem PicoC-Compiler.

PicoC

Funktionalitäten

Die Sprache L_{PicoC} ist eine **Untermenge** der Sprache L_C , welche:

- ▶ **Einzeilige Kommentare** `//` und **Mehrzeilige Kommentare** `/* comment */`.
- ▶ die **Basisdatentypen** `int`, `char` und `void`.
- ▶ die **Zusammengesetzten** Datentypen **Felder** (z.B. `int ar[3]`), **Verbunde** (z.B. `struct st {int attr1; int attr2;}`) und **Zeiger** (z.B. `int *pntr`), inklusive:
 - ▶ **Initialisierung** (z.B. `struct st st_var = {.attr1=42, .attr2={.attr={&var, &var}}}`).
 - ▶ dazugehörige **Operationen** `[i]`, `.attr`, `*` und `&`.

PicoC

Funktionalitäten

- ▶ die **Zusammengesetzten** Datentypen **Felder** (z.B. `int ar[3]`), **Verbunde** (z.B. `struct st {int attr1; int attr2;}`) und **Zeiger** (z.B. `int *pntr`), inklusive:
 - ▶ **Kombinationen** der eben genannten **Operationen** (z.B. `(*complex_var[0][1])[1].attr`) und **Datentypen** (z.B. `struct st (*complex_var[1][2])[2]`).
 - ▶ **Zeigerarithmetik** (z.B. `*(var + 2)`).
- ▶ `if(cond){ }-` und `else{ }-`**Anweisungen**, inklusive:
 - ▶ Kombination von `if` und `else`, nämlich `else if(cond){ }`.
- ▶ `while(cond){ }-` und `do while(cond){ };`**Anweisungen**.

PicoC

Funktionalitäten

- ▶ **Arihmetische und Bitweise Ausdrücke**, welche mithilfe der **binären Operatoren** `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `<<`, `>>` und **unären Operatoren** `-`, `~` umgesetzt sind.
- ▶ **Logische Ausdrücke**, welche mithilfe der **Relationen** `==`, `!=`, `<`, `>`, `<=`, `>=` und **Logischer Verknüpfungen** `!`, `&&`, `||` umgesetzt sind.
- ▶ **Zuweisungen**, welche mithilfe des **Zuweisungsoperators** `=` umgesetzt sind, inklusive:
 - ▶ Zuweisung an **Feldelement**, **Verbundsattribut** oder **Zeigerelement** (z.B. `(*var.attr)[2] = fun() + 42`).

PicoC

Funktionalitäten

- ▶ **Funktionsdefinitionen** (z.B. `int fun(int arg1[3], struct st arg2){}`), inklusive:
 - ▶ **Funktionsdeklarationen** (z.B. `int fun(int arg1[3], struct st arg2);`).
 - ▶ **Funktionsaufrufe** (z.B. `fun(ar, st_var)`)
 - ▶ **Sichtbarkeitsbereiche** innerhalb der Codeblöcke `{}` der Funktionen.
 - ▶ **Argumentübergabe** erfolgt ausschließlich über die **Call-by-Value** Strategie.
 - ▶ bei **Feldern** wird ein **Zeiger** in den Stackframe der **aufrufenden Funktion** geschrieben.
 - ▶ bei **Verbunden** wird der **komplette Verbund** in den Stackframe der **aufrufenden Funktion** **kopiert**.

PicoC

Sonstiges

- ▶ Implementierung ist aufgebaut auf **RETI-Codeschnipseln** aus der Vorlesung Scholl, „**Betriebssysteme**“, Kapitel 3 Übersetzung höherer Programmiersprachen in Maschinensprache.
 - ▶ bei **Inkonsistenzen** und **Umstimmigkeiten** angepasst.
- ▶ im Appendix ab Folie ?? weitere Informationen.

RETI-Architektur

- **32-Bit Architektur**, die in den Vorlesungen Scholl, „Betriebssysteme“ und Scholl, „Technische Informatik“ zu Lernzwecken eingesetzt wird. Basis für L_{RETI} .

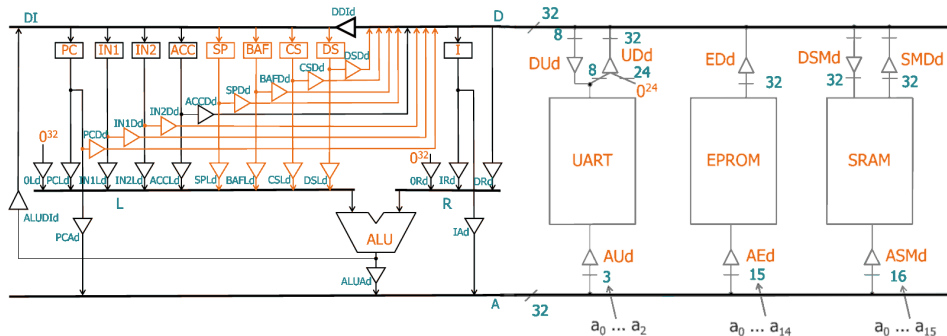


Abbildung 3: Datenpfade der RETI-Architektur, nicht selbst erstellt, leicht abgeändert.

RETI-Architektur

Speicherorganisation

- Register haben bestimmte Aufgaben bei der Umsetzung von Prozessen.

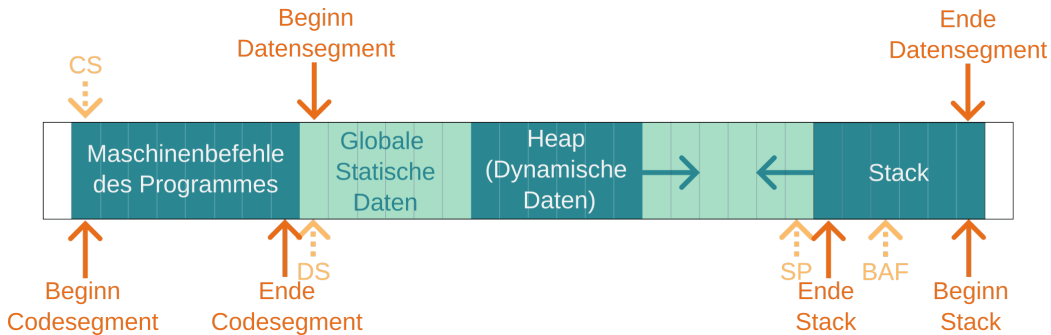


Abbildung 4: Speicherorganisation.

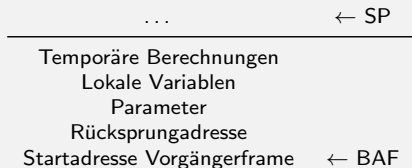
Speicherorganisation

Lokale Variablen und Parameter von Funktionen

- ▶ alle Funktionen, **außer** der `main`-Funktion besitzen einen **Stackframe** für **Lokale Variablen** und **Parameter**.
- ▶ **Globale Statische Daten** sind **Globale Variablen**, sowie **Lokale Variablen** und **Parameter** der `main`-Funktion.

Stackframe

- ▶ Datenstruktur, um **Zustand** einer Funktion zur **Laufzeit** zu „konservieren“.



Implementierung

Lexikalische Analyse

Aufgabe

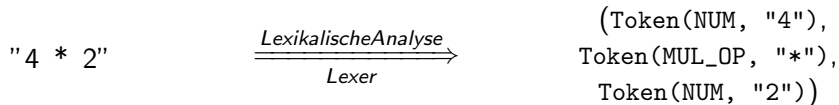


Abbildung 5: Aus Eingabewort Tokens generieren.

- ▶ im Appendix ab Folie ?? genauer erklärt.

Syntaktische Analyse

Aufgabe

`(Token(NUM, "4"),
Token(MUL_OP, "*"),
Token(NUM, "2"))`

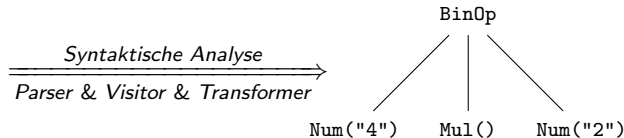


Abbildung 6: Aus Tokens einen Abstrakten Syntaxbaum generieren.

- ▶ **Parser** generiert im Zusammenspiel mit **Lexer** den **Ableitungsbaum**.
- ▶ **Visitor** vereinfacht den **Ableitungsbaum**.
- ▶ **Transformer** generiert den **Abstrakten Syntaxbaum**.
- ▶ im Appendix ab Folie ?? genauer erklärt.

Syntaktische Analyse

Zwischenschritte

"4 * 2"

Parser
 $\xrightarrow{\hspace{1cm}}$
Lexer

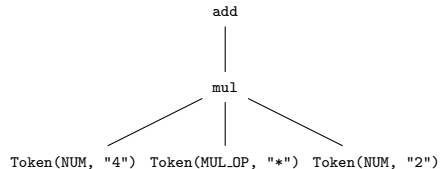
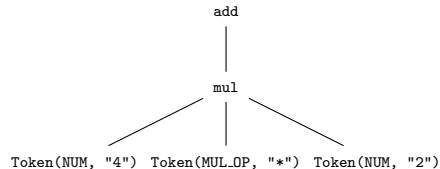


Abbildung 7: Aus dem Eingebewort einen Ableitungsbaum generieren.



Visitor
 $\xrightarrow{\hspace{1cm}}$
Transformer

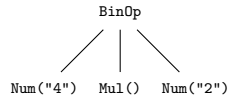


Abbildung 8: Aus Ableitungsbaum Abstrakten Syntaxbaum generieren.

Syntaktische Analyse

Lark Parsing Toolkit

- ▶ erleichtert **Lexikalische Analyse** und **Syntaktische Analyse**.
- ▶ **Basic Lexer**, **Earley Parser**, **Visitor** und **Transformer** implementiert.

NUM	::=	"4" "2"	<i>L_Lex</i>
ADD_OP	::=	"+"	
MUL_OP	::=	"*"	
mul	::=	<i>mul</i> MUL_OP NUM NUM	<i>L_Parse</i>
add	::=	<i>add</i> ADD_OP mul mul	

Grammatik 1: Grammatik für Lexer oben und Grammatik für Parser unten.

Code Generierung

Aufgabe

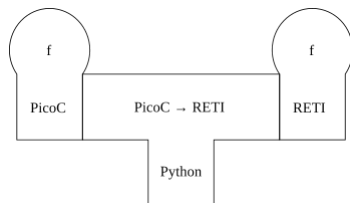


Abbildung 9: T-Diagramm für die Aufgabe der Code Generierung.

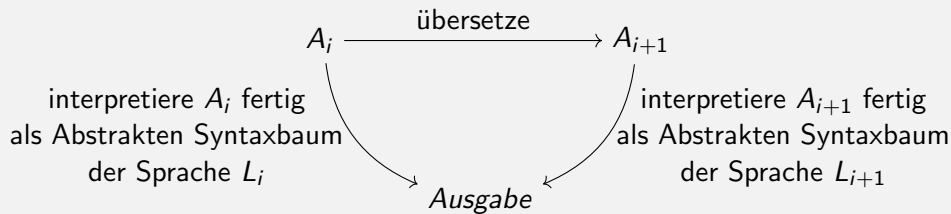
- ▶ Abstrakter Syntaxbaum für Sprache L_{PicoC} soll zu Abstraktem Syntaxbaum der Sprache L_{RETI} umgeformt werden.
- ▶ mit Passes kleinschrittig immer mehr der Syntax der Maschinensprache annähern.

Code Generierung

Definitionen

Pass

- ▶ Übersetzungsschritt eines Abstrakten Syntaxbaumes von L_i zu L_{i+1}
- ▶ beide Abstrakten Syntaxbäume gleiche Semantik
- ▶ übernimmt Teilaufgabe, keine Überschneidung



Code Generierung

Überblick über Passes des PicoC-Compilers

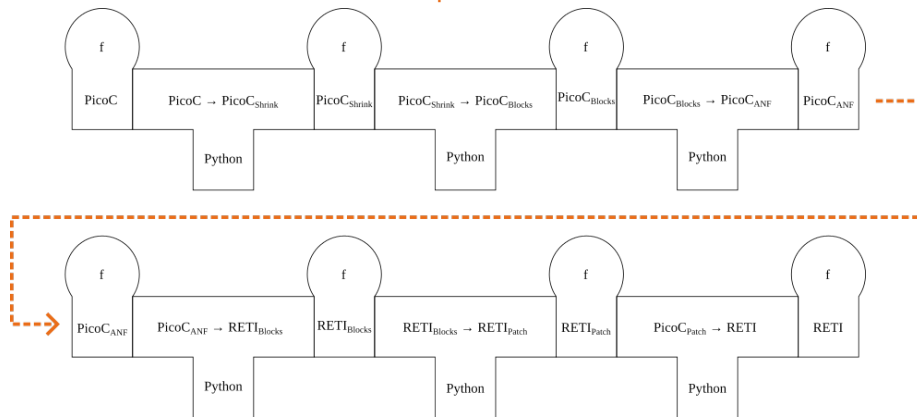
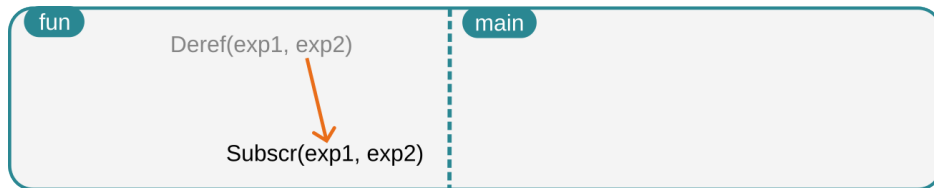


Abbildung 10: Architektur mit allen Passes ausgeschrieben.

Code Generierung

PicoC-Shrink Pass

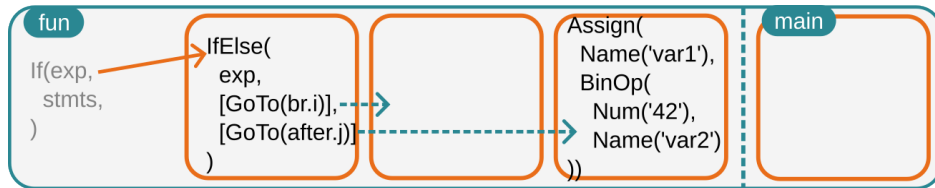
- ▶ gleiche Semantik des Dereferenzierungsoperators `*(pntr_or_ar + i)` und des Operators für Indexzugriff auf ein Feld `pntr_or_ar[i]`, sind austauschbar.
- ▶ Ersetzen von `Deref(exp, i)` durch `Subscr(exp, i)`.
- ▶ Dereferenzierung `*(pntr_or_ar + i)` wird von den Routinen für einen Indexzugriff auf ein Feld `pntr_or_ar[i]` übernommen \Rightarrow kein redundanter Code.



Code Generierung

PicoC-Blocks Pass

- ▶ If(exp, stmts), IfElse(exp, stmts1, stmts2), While(exp, stmts) und DoWhile(exp, stmts) durch Block(name, stmts_instrs-, GoTo(lable)- und IfElse(exp, stmts1, stmts2) ersetzt.
- ▶ im Appendix ab Folie ?? genauer erklärt.



Code Generierung

PicoC-ANF Pass

- ▶ formt **Abstrakten Syntaxbaum** um, sodass er die **Syntax** der Sprache L_{PicoC_ANF} erfüllt, deren Grammatik in **A-Normalform** ist.
- ▶ **Funktionen** werden aufgelöst.
- ▶ im Appendix ab Folie ?? genauer erklärt.

```
Exp(Global(Num('addr1'))),  
IfElse(Stack(Num('1'),  
  [GoTo(br.i)],  
  [GoTo(after.j)])  
)
```

```
Exp(Num('42')),  
Exp(Global(Num('addr1'))),  
BinOp(Stack(Num('2')), Mul(), Stack(Num('1'))),  
Assign(Global(Num('addr2')), Stack(Num('1'))),
```

PicoC-ANF Pass

A-Normalform

- ▶ **Zweck:** Maschinenbefehlen annähern, die meist nur eine Aktion ausführen. Eine Anweisung wird aufgespalten, wenn sie mehreren Aktionen entspricht. Nebeneffekte, die den Kompiliervorgang beeinflussen werden isoliert.
- ▶ im Appendix ab Folie ?? genauer erklärt.

Code

```
void main() {  
    int x = 1 - 5 * 4;  
}
```

ziehe **Komplexe Ausdrücke** aus
Anweisungen und Ausdrücken **vor**

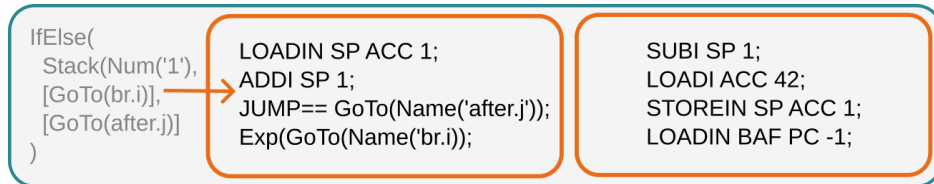
Code in A-Normalform

```
void main() {  
    int x; // allocate at address  
    ↪ 0 relative to DS Register  
    1;  
    5;  
    4;  
    stack(2) * stack(1);  
    stack(2) - stack(1);  
    global(0) = stack(1);  
}
```


Code Generierung

RETI-Blocks Pass

- **PicoC-Knoten**, die **Anweisungen** darstellen, werden durch **semantisch** entsprechende **RETI-Knoten**, die **Befehle** darstellen ersetzt.



Code Generierung

RETI-Patch Pass

- ▶ **Ausbessern** (engl. to patch) des **Abstrakten Syntaxbaumes** durch:
 - ▶ **Einfügen** eines `GoTo(Name('main'))` in den `global.<number>`-Block, wenn `main`-Funktion **nicht** die **erste Funktion** ist.
 - ▶ **Entfernen** von `GoTo()`s, deren Sprung nur **eine** Adresse weiterspringt.
 - ▶ weitere Aufgaben im Appendix ab Folie ?? aufgezählt.

```
Exp( global
  GoTo(Name(
    'main.k')
  )
)
```


```
LOADIN SP ACC 1;
ADDI SP 1;
JUMP== GoTo(Name('after.j'));
Exp(GoTo(Name('br.i'));
```

```
LOADIN SP ACC 1;
ADDI SP 1;
JUMP== GoTo(Name('after.j'));
# // not included
```

Code Generierung

RETI Pass

- ▶ verbliebene **PicoC-Knoten** werden durch entsprechende **RETI-Knoten** ersetzt:
 - ▶ keine **Blöcke** mehr, Knoten genauso **zusammengefügt**, wie sie in diesen **angeordnet** waren.
 - ▶ `GoTo(Name(str))` werden durch einen **Immediate** mit passender **Distanz** / **Adresse** oder einen **Sprungbefehl** mit passender **Distanz** `Jump(Always(), Im(str(distance)))` ersetzt.

<pre>Exp(GoTo(Name('main.k'))) JUMP <distance-to-main.k></pre>	<pre>LOADIN SP ACC 1; ADDI SP 1; JUMP== GoTo(Name('after.j')); # // not included</pre>		<pre>LOADIN SP ACC 1; ADDI SP 1; JUMP== <distance-to-after.j>; # // not included</pre>
--	--	--	--

Code Generierung

Codebeispiel

- ▶ im Appendix am Folie ?? sind Tokens, Ableitungsbaum, Vereinfachter Ableitungsbaum, Abstrakter Syntaxbaum und die modifizierten Abstrakten Synntaxbäume der verschiedenen Passes an einem Codebeispiel erklärt.
- ▶ `> make test TESTNAME=example_presentation VERBOSE=-v`

Code Generierung

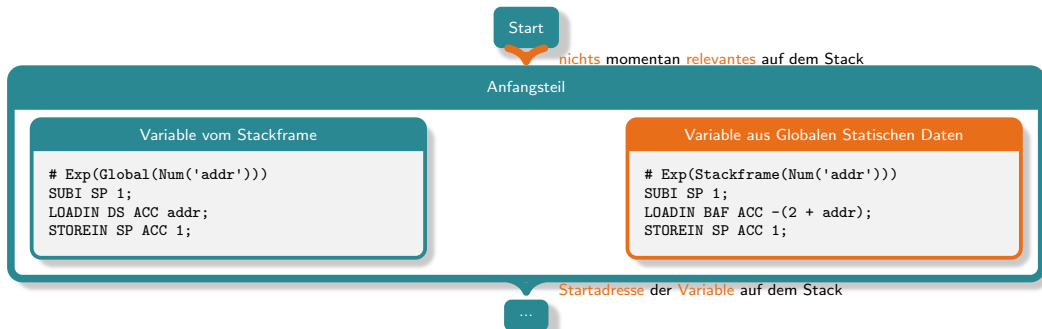
Zugriff auf Zusammengesetzte Datentypen

```
1 // in:1
2 // expected:42
3 // datasegment:36
4
5 struct stt {int attr1; int attr2[2];};
6
7 struct stt ar_of_sts[3][2];
8
9 int fun(struct stt (*param)[3][2]){
10     ((*param+2))[1].attr2[input()] = 42;
11     return 1;
12 }
13
14 void main() {
15     struct stt (*pntr_on_ar_of_sts)[3][2] = &ar_of_sts;
16     int res = fun(pntr_on_ar_of_sts);
17     if (res) {
18         print(((*pntr_on_ar_of_sts+2))[1].attr2[1]);
19     }
20 }
```

Code Generierung

Codebeispiel

► `(*(*pntr_on_ar_of_sts+2))[1].attr2[1]`



► `(*(*pntr_on_ar_of_sts+2))[1].attr2[1]`  `struct stt (*pntr_on_ar_of_sts)[3][2]...`

 Startadresse der Variable auf dem Stack

Mittelteil

Feldindexzugriff auf Feld

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN1 2;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Feldindexzugriff auf Zeiger

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN2 2;
LOADIN IN2 IN1 0;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Verbundsattribut-zugriff auf Verbund

```
# Ref(Attr(Stack(Num('1')),
↪ Name('attr')))
LOADIN SP IN1 1;
ADDI IN1  $\sum_{k=1}^{\text{idx}-1} \text{size}(\text{datatype}_{1,k})$ ;
STOREIN SP IN1 1;
```

*1

 *1: Startadresse eines Zeigerelementes, Feldelementes oder Verbundsattributes auf dem Stack

► `(*(*pntr_on_ar_of_sts+2))[1].attr2[1]`  `struct stt (*pntr_on_ar_of_sts)[3][2]...`

Startadresse der Variable auf dem Stack

Mittelteil

Feldindexzugriff auf Feld

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN1 2;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Feldindexzugriff auf Zeiger

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN2 2;
LOADIN IN2 IN1 0;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Verbundsattribut-zugriff auf Verbund

```
# Ref(Attr(Stack(Num('1')),
↪ Name('attr')))
LOADIN SP IN1 1;
ADDI IN1  $\sum_{k=1}^{\text{idx}-1} \text{size}(\text{datatype}_{1,k})$ ;
STOREIN SP IN1 1;
```

*1

*1: Startadresse eines Zeigerelementes, Feldelementes
oder Verbundsattributes auf dem Stack

► `(*(pntr_on_ar_of_sts+2))[1].attr2[1]`  `struct stt (pntr_on_ar_of_sts)[3][2]...`

Startadresse der Variable auf dem Stack

Mittelteil

Feldindexzugriff auf Feld

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN1 2;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Feldindexzugriff auf Zeiger

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN2 2;
LOADIN IN2 IN1 0;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Verbundsattribut-zugriff auf Verbund

```
# Ref(Attr(Stack(Num('1')),
↪ Name('attr')))
LOADIN SP IN1 1;
ADDI IN1  $\sum_{k=1}^{\text{idx}-1} \text{size}(\text{datatype}_{1,k})$ ;
STOREIN SP IN1 1;
```

*1

*1: Startadresse eines Zeigerelementes, Feldelementes
oder Verbundsattributes auf dem Stack

► `(*(pntr_on_ar_of_sts+2))[1].attr2[1]` `struct stt (pntr_on_ar_of_sts)[3][2]...`

Startadresse der Variable auf dem Stack

Mittelteil

Feldindexzugriff auf Feld

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN1 2;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Feldindexzugriff auf Zeiger

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN2 2;
LOADIN IN2 IN1 0;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Verbundsattribut-zugriff auf Verbund

```
# Ref(Attr(Stack(Num('1')),
↪ Name('attr')))
LOADIN SP IN1 1;
ADDI IN1  $\sum_{k=1}^{\text{idx}-1} \text{size}(\text{datatype}_{1,k})$ ;
STOREIN SP IN1 1;
```

*1

*1: Startadresse eines Zeigerelementes, Feldelementes
oder Verbundsattributes auf dem Stack

► `(*(*pntr_on_ar_of_sts+2))[1].attr2[1],` `struct stt {int attr1; int attr2[2];};`



Startadresse der Variable auf dem Stack

Mittelteil

Feldindexzugriff auf Feld

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN1 2;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Feldindexzugriff auf Zeiger

```
# z.B. Exp(Num('idx'))
SUBI SP 1;
LOADI ACC idx;
STOREIN SP ACC 1;
# Ref(Subscr(Stack(Num('2')),
↪ Stack(Num('1'))))
LOADIN SP IN2 2;
LOADIN IN2 IN1 0;
LOADIN SP IN2 1;
MULTI IN2  $(\prod_{j=i+1}^n \text{din}_j) \cdot \text{size}(\text{datatype})$ ;
ADD IN1 IN2;
ADDI SP 1;
STOREIN SP IN1 1;
```

Verbundsattribut-zugriff auf Verbund

```
# Ref(Attr(Stack(Num('1')),
↪ Name('attr')))
LOADIN SP IN1 1;
ADDI IN1  $\sum_{k=1}^{\text{idx}-1} \text{size}(\text{datatype}_{1,k})$ ;
STOREIN SP IN1 1;
```



*1

*1: Startadresse eines Zeigerelementes, Feldelementes
oder Verbundsattributes auf dem Stack



► `(*(pnter_on_ar_of_sts+2))[1].attr2[1],` `struct stt {int attr1; int attr2[2];};`

...

Startadresse eines Zeigerelementes, Feldelementes
oder Verbundattributes auf dem Stack

Schlussteil

Letzter Datentyp ist Verbund,
Zeiger oder Basisdatentyp

```
# Exp(Stack(Num('1')))
LOADIN SP IN1 1;
LOADIN IN1 ACC 0;
STOREIN SP ACC 1;
```

Letzter Datentyp ist Feld

```
# not included Exp(Stack(Num('1')))
```

Inhalt der Speicherzelle an der berechneten
Adresse oder die berechnete Adresse selbst

Ende

Fehlermeldungen

Kategorien

- ▶ UnexpectedCharacter
- ▶ UnexpectedToken
- ▶ UnexpectedEOF
- ▶ DivisionByZero
- ▶ UnknownIdentifier
- ▶ UnknownAttribute
- ▶ ReDeclarationOrDefinition
- ▶ TooLargeLiteral
- ▶ NoMainFunction
- ▶ ConstAssign
- ▶ DatatypeMismatch
- ▶ PrototypeMismatch
- ▶ ArgumentMismatch
- ▶ WrongNumberArguments
- ▶ WrongReturnType
- ▶ im Appendix ab Folie ?? mit Erklärung.

Qualitätssicherung

Tests

Typischer Test

```
// in:21 2 6 7
// expected:42 42
// datasegment:4
```

```
void main() {
    print(input() * input());
    print(input() * input());
}
```

convert_to_c.py →

```
#include<stdio.h>
```

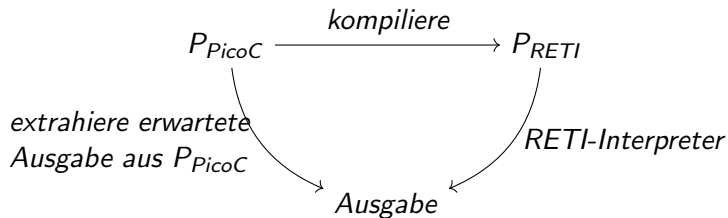
```
void main() {
    printf(" %d", 21 * 2);
    printf(" %d", 6 * 7);
}
```

- ▶ **Eingaben** // in:<space-sep-values> in <program>.in
- ▶ **Erw. Ausgaben** // expected:<space-sep-values> in <program>.out_expected.
- ▶ **Datensegmentgröße** // datasegment:<size> in <program>.datasegment_size.
- ▶ **jede Ausgabe** eines print(exp) wird in <program>.out geschrieben.
- ▶ **jede Ausgabe** eines printf("%d", exp) wird in <program>.c_out geschrieben.

Tests

Ablauf

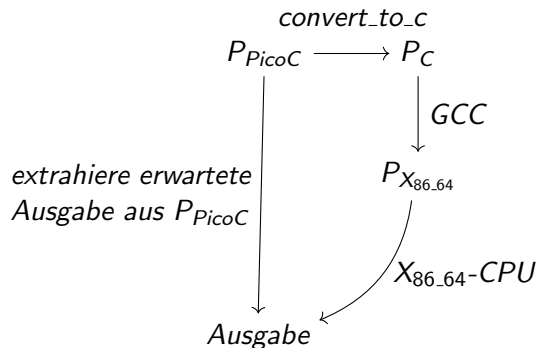
- ▶ prüfen, ob **Inhalt** von `<program>.out_expected` und `<program>.out` **identisch**.
- ▶ **Schreiber** der Tests = **Implementierer** des PicoC-Compiler \Rightarrow Tests **bestätigen** nur das PicoC-Compiler genauso implementiert, wie diese Person die **Semantik** von L_{PicoC} **interpretiert** hat.



Tests

Ablauf

- ▶ der **GCC** setzt die **Semantik** von L_{PicoC} sehr wahrscheinlich korrekt um.
- ▶ prüfen, ob **Inhalt** von `<program>.out_expected` und `<program>.c_out` **identisch**.



Tests

Durchlauf aller Tests

- `> make test <more-options>` (siehe Appendix auf Folie ??)

```
> make test
=====
= ./tests/basic_array_init.picoc =
=====
...
=====
=          Verification          =
=====
./tests/basic_array_init.c
...
=====
=          Results              =
=====
Verified: 104 / 104
Not verified:
Running through: 180 / 180
Not running through:
Passed: 180 / 180
Not passed:
```

Tests

Sonstiges

- ▶ <https://github.com/matthejue/PicoC-Compiler>.
- ▶ im Appendix auf Folie ?? sind die verschiedenen **Bedienmöglichkeiten** zum Ausführen der Tests erklärt.
- ▶ im Appendix auf Folie ?? sind die verschiedenen **Testkategorien** mit Erklärung zu finden.

Vorführung

Bedienung

Wichtigste Funktionalitäten

- ▶ **Kompilieren:** `> picoc_compiler <cli-options> program.picoc .`
 - ▶ alle `<cli-options>` im Appendix ab Folie ?? aufgelistet.
- ▶ **Kompilieren + Interpretieren:**
`> picoc_compiler <cli-options> -R program.picoc .`
- ▶ **Shell-Mode:** `> picoc_compiler` ohne Argumente.
 - ▶ alle Befehle des Shell-Modes im Appendix ab Folie ?? aufgelistet.
- ▶ **Show-Mode:**
 - ▶ **Bedienung** des Show-Modes im Appendix ab Folie ?? erklärt.

Bedienung

Shell-Mode

```
> picoc_compiler
PicoC Shell. Enter `help` (shortcut `?`) to see the manual.
PicoC> cpl "6 * 7;";
----- RETI -----
SUBI SP 1;
LOADI ACC 6;
STOREIN SP ACC 1;
SUBI SP 1;
LOADI ACC 7;
STOREIN SP ACC 1;
LOADIN SP ACC 2;
LOADIN SP IN2 1;
MULT ACC IN2;
STOREIN SP ACC 2;
ADDI SP 1;
LOADIN BAF PC -1;

Compilation successfull

PicoC> quit
```

Code 1: Shell-Mode und die Befehle `compile` und `quit`.

Bedienung

Shell-Mode

```
PicoC> mu "int var = 42;";
----- Code -----
// stdin.picoc:
void main() {int var = 42;}
----- Tokens -----
...
----- Abstract Syntax Tree -----
...
----- PicoC Shrink -----
...
----- RETI -----
SUBI SP 1;
LOADI ACC 42;
STOREIN SP ACC 1;
LOADIN SP ACC 1;
STOREIN DS ACC 0;
ADDI SP 1;
LOADIN BAF PC -1;
----- RETI Run -----
...

Compilation successfull
```

Code 2: Shell-Mode und der Befehl most used.

Bedienung

Show-Mode

```
▶ > make test-show TESTNAME=<testname> PAGES=<pages>
```

Index:	04	00019 ADD ACC C5;	00057 LOADIN SP ACC 2;	00095 MOVE BAF ACC;	00130 ?
Instruction:	STOREIN SP ACC 2;	00020 STOREIN BAF ACC -1;	00058 LOADIN SP IN2 1;	00096 ADDI SP 3;	00134 0
ACC:	42	00021 JUMP 44;	00059 ADD ACC IN2;	00097 MOVE SP BAF;	00135 0
ACC_SIMPLE:	42	00022 MOVE BAF IN1;	00060 STOREIN SP ACC 2;	00098 SUBI SP 4;	00136 0
IN1:	0	00023 LOADIN IN1 BAF 0;	00061 ADDI SP 1; < PC	00099 STOREIN BAF ACC 0;	00137 0
IN1_SIMPLE:	0	00024 MOVE IN1 SP;	00062 LOADIN SP ACC 1;	00100 LOADI ACC 103;	00138 0 < SP
IN2:	2	00025 SUBI SP 1;	00063 ADDI SP 1;	00101 ADD ACC C5;	00139 2
IN2_SIMPLE:	2	00026 STOREIN SP ACC 1;	00064 LOADIN BAF PC -1;	00102 STOREIN BAF ACC -1;	00140 42
PC:	2147483700	00027 LOADIN SP ACC 1;	00065 SUBI SP 1;	00103 JUMP -50;	00141 2
PC_SIMPLE:	61	00028 STOREIN DS ACC 1;	00066 LOADI ACC 2;	00104 MOVE BAF IN1;	00142 40
SP:	2147483786	00029 ADDI SP 1;	00067 STOREIN SP ACC 1;	00105 LOADIN IN1 BAF 0;	00143 2147483752
SP_SIMPLE:	138	00030 SUBI SP 1;	00068 STOREIN SP ACC 1;	00106 MOVE IN1 SP;	00144 2147483797 < BAF
BAF:	2147483792	00031 LOADIN DS ACC 1;	00069 LOADIN BAF ACC -3;	00107 SUBI SP 1;	00145 40
BAF_SIMPLE:	144	00032 STOREIN SP ACC 1;	00070 ADDI SP 1;	00108 STOREIN SP ACC 1;	00146 2
CS:	2147483651	00033 SUBI SP 1;	00071 SUBI SP 1;	00109 LOADIN SP ACC 1;	00147 38
CS_SIMPLE:	3	00034 LOADI ACC 2;	00072 LOADIN BAF ACC -2;	00110 ADDI SP 1;	00148 2147483678
CS_SIMPLE:	2147483766	00035 STOREIN SP ACC 1;	00073 STOREIN SP ACC 1;	00111 CALL PRINT ACC;	00149 2147483658
DS:	118	00036 LOADIN SP ACC 2;	00074 SUBI SP 1;	00112 SUBI SP 1;	00150 0
DS_SIMPLE:	118	00037 LOADIN SP IN2 1;	00075 LOADIN BAF ACC -3;	00113 LOADIN BAF ACC -4;	00000 0
DSIM:		00038 ADD ACC IN2;	00076 STOREIN SP ACC 1;	00114 STOREIN SP ACC 1;	00001 0
00000 JUMP 0;		00039 STOREIN SP ACC 2;	00077 LOADIN SP ACC 2;	00115 LOADIN SP ACC 1;	00002 0
00001 2147483648		00040 ADDI SP 1;	00078 LOADIN SP IN2 1;	00116 ADDI SP 1;	00003 0
00002 0		00041 LOADIN SP ACC 1;	00079 ADDI ACC IN2;	00117 LOADIN BAF PC -1;	00004 0
00003 CALL INPUT ACC; < CS		00042 ADDI SP 1;	00080 STOREIN SP ACC 2;	00118 38 < DS	00005 0
00004 SUBI SP 1;		00043 CALL PRINT ACC;	00081 ADDI SP 1;	00119 0	00006 0
00005 STOREIN SP ACC 1;		00044 LOADIN BAF PC -1;	00082 LOADIN SP ACC 1;	00120 0	00007 0
00006 LOADIN SP ACC 1;		00045 SUBI SP 1;	00083 STOREIN BAF ACC -4;	00121 0	00008 0
00007 STOREIN DS ACC 0;		00046 LOADI ACC 2;	00084 ADDI SP 1;	00122 0	00009 0
00008 ADDI SP 1;		00047 STOREIN SP ACC 1;	00085 SUBI SP 1;	00123 0	00010 0
00009 SUBI SP 2;		00048 LOADIN SP ACC 1;	00086 LOADIN BAF ACC -4;	00124 0	00011 0
00010 SUBI SP 1;		00049 STOREIN BAF ACC -3;	00087 STOREIN SP ACC 1;	00125 0	00012 0
00011 LOADIN DS ACC 0;		00050 ADDI SP 1;	00088 LOADIN SP ACC 1;	00126 0	00013 0
00012 STOREIN SP ACC 1;		00051 SUBI SP 1;	00089 ADDI SP 1;	00127 0	00014 0
00013 MOVE BAF ACC;		00052 LOADIN BAF ACC -2;	00090 CALL PRINT ACC;	00128 0	00015 0
00014 ADDI SP 3;		00053 STOREIN SP ACC 1;	00091 SUBI SP 1;	00129 0	00016 0
00015 MOVE SP BAF;		00054 SUBI SP 1;	00092 SUBI SP 1;	00130 0	00017 0
00016 SUBI SP 5;		00055 LOADIN BAF ACC -3;	00093 LOADIN BAF ACC -4;	00131 0	00018 0
00017 STOREIN BAF ACC 0;		00056 STOREIN SP ACC 1;	00094 STOREIN SP ACC 1;	00132 0	00019 0
00018 LOADI ACC 191;					

Abbildung 11: Show-Mode in der Verwendung.

Codebeispiel 1

Finbonacci

```
1 // in:10
2 // expected:55
3 // datasegment:64
4 // from the Operating Systems Lecture by Prof. Dr. Christoph Scholl
5
6 int ar[11] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
7
8 int fib_efficient(int n, int* res){
9     if (n == 0)
10         return 0;
11     else if (n == 1){
12         res[0] = 0;
13         res[1] = 1;
14         return 1;
15     }
16     res[n] = fib_efficient(n - 1, res) + res[n - 2];
17     return res[n];
18 }
19
20 void main() {
21     print(fib_efficient(input(), ar));
22 }
```

Codebeispiel 2

Bubble Sort

```
1 // in:0 5
2 // expected:-2 314
3 // based on a function from the Operating Systems Lecture by Prof. Dr. Christoph Scholl and
  ↳ https://de.wikipedia.org/wiki/Bubblesort
4
5 struct stt {int len; int *ar;};
6
7 int ar[6] = {314, 42, 4, 42, -2, 5};
8
9 struct stt st_ar = {.len=6, .ar=ar};
10
11 int swap(int *x, int *y) {
12     // in the lecture this function is called pairsort
13     int h;
14     int swapped = 0;
15     if (*x > *y) {
16         h = *x; *x = *y; *y = h; swapped = 1;
17     }
18     return swapped;
19 }
```

Codebeispiel 2

Bubble Sort, Teil 2

```
1 void main() {  
2     int swapped;  
3     int i;  
4     int n = st_ar.len-1;  
5     do {  
6         i = 0;  
7         while (i < n) {  
8             swapped = swap(&st_ar.ar[i], &st_ar.ar[i+1]);  
9             i = i + 1;  
10        }  
11        n = n - 1;  
12    } while(swapped);  
13    print(st_ar.ar[input()]);  
14    print(st_ar.ar[input()]);  
15 }
```

Codebeispiel 3

Min Sort

```
1 // in:
2 // expected:-2 4 5 42 42 314
3 // from the Algorithms and Datastructures Lecture by Prof. Dr. Bast
4
5 struct stt {int len; int *ar;};
6
7 void min_sort(int *ar, int len) {
8     int i = 0;
9     int j;
10    int minimum;
11    int minimum_index;
12    int tmp;
13    while (i < len) {
14        minimum = ar[i];
15        minimum_index = i;
16        j = i + 1;
17        while (j < len) {
18            if (ar[j] < minimum) {
19                minimum = ar[j];
20                minimum_index = j;
21            }
22            j = j + 1;
23        }
```

Codebeispiel 3

Min Sort, Teil 2

```
1     tmp = ar[i];
2     ar[i] = ar[minimum_index];
3     ar[minimum_index] = tmp;
4     i = i + 1;
5 }
6 }
7
8 void main() {
9     int len = 6;
10    int ar[6] = {314, 42, 4, 42, -2, 5};
11    min_sort(ar, len);
12    print(ar[0]);
13    print(ar[1]);
14    print(ar[2]);
15    print(ar[3]);
16    print(ar[4]);
17    print(ar[5]);
18 }
```

Codebeispiel 4

Fakultät

```
1 // in:3 4
2 // expected:6 24
3 // from the Operating Systems Lecture by Prof. Dr. Christoph Scholl
4
5 int fakul(int n) {
6     int res_f; int h;
7     if (n == 1) {
8         res_f = 1;
9     } else {
10         h = fakul(n-1);
11         res_f = n * h;
12     }
13     return res_f;
14 }
15
16 void main() {
17     int res;
18     print(fakul(input()));
19     res = fakul(input());
20     print(res);
21 }
```

Codebeispiel 5

Binary Search

```
1 // in:41 42
2 // expected:-1 5
3 // datasegment:64
4 // from the Introduction to Programming Lecture by Peter Thiemann
5
6 struct ar_with_lent {int len; int *ar;};
7
8 int ar[10] = {1, 3, 4, 7, 19, 42, 128, 314, 512, 1024};
9
10 struct ar_with_lent ar_with_len = {.len=10, .ar=ar};
11
12 int bsearch_rec(int *ar, int key, int lo, int hi) {
13     int m;
14     if (lo == hi)
15         return -1; // key not in empty segment
16     m = (lo + hi) / 2; // position of root
17     if (ar[m] == key)
18         return m;
19     else if (ar[m] > key)
20         return bsearch_rec(ar, key, lo, m);
21     else // ar[m] < key
22         return bsearch_rec(ar, key, m+1, hi);
23 }
```

Codebeispiel 5

Binary Search, Teil 2

```
1 void main() {  
2     print(bsearch_rec(ar_with_len.ar, input(), 0, ar_with_len.len - 1));  
3     print(bsearch_rec(ar_with_len.ar, input(), 0, ar_with_len.len - 1));  
4 }
```


Codebeispiel 6

Primzahlen bis Zahl n

```
1 // in:30
2 // expected:2 3 5 7 11 13 17 19 23 29
3 // from the Introduction to Programming Lecture by Peter Thiemann
4
5 int ar[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
6 int len = 10;
7
8 void primes(int *primes, int n) {
9     int i = 3;
10    int j;
11    int idx = 1;
12    char undividable = 1;
13    if (n <= 1)
14        return;
15    primes[0] = 2;
16
17    while (i <= n) {
18        j = 0;
19        while (j < idx) {
20            if (i % primes[j] == 0) {
21                undividable = 0;
22            }
23            j = j + 1;
24        }
```

Codebeispiel 6

Primzahlen bis Zahl n, Teil 2

```
1     if (undividable) {
2         primes[idx] = i;
3         idx = idx + 1;
4     }
5     undividable = 1;
6     i = i + 1;
7 }
8 }
9
10 void main() {
11     int i = 0;
12     primes(ar, input());
13     while (i < len) {
14         print(ar[i]);
15         i = i + 1;
16     }
17 }
```

Bedienung

Tutorials

- ▶ https://github.com/matthejue/PicoC-Compiler/blob/new_architecture/doc/getting_started.md.

Literatur

Vorlesungen



Scholl, Christoph. „Betriebssysteme“. Vorlesung. Vorlesung. Universität Freiburg, 2020. URL:

https://abs.informatik.uni-freiburg.de/src/teach_main.php?id=157
(besucht am 09.07.2022).



— „Technische Informatik“. Vorlesung. Vorlesung. Universität Freiburg, 3. Aug. 2022. (Besucht am 03.08.2022).

Online