

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

28. September 2022

Universität Freiburg, Lehrstuhl für Betriebssysteme

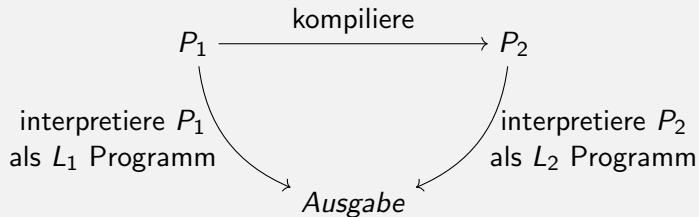
Einführung

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 .`

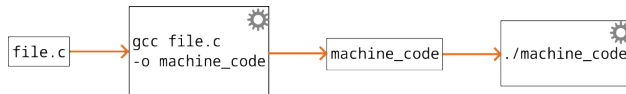


Abbildung 1: Schritte zum Ausführen eines Programmes mit dem GCC.

- 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 **61** weitere Informationen.

- [illegible]

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

RETI-Architektur

Speicherorganisation

- Register haben bestimmte Aufgaben bei der Umsetzung von Prozessen.

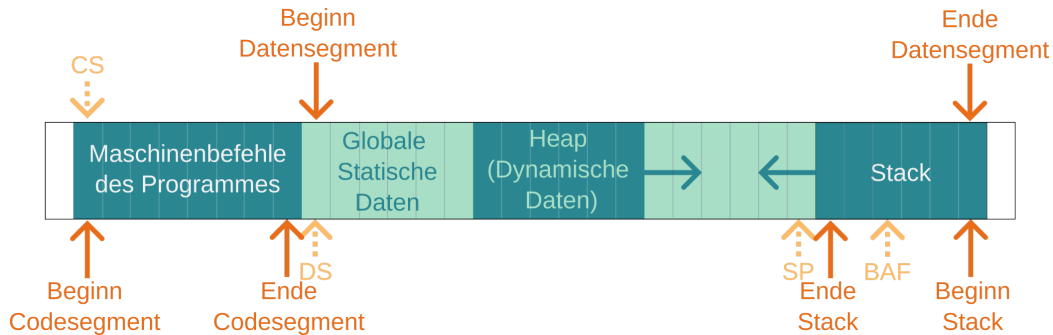


Abbildung 4: Speicherorganisation.

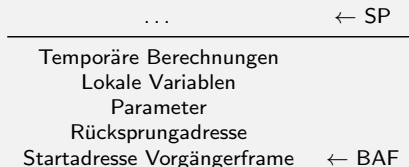
Speicherorganisation

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

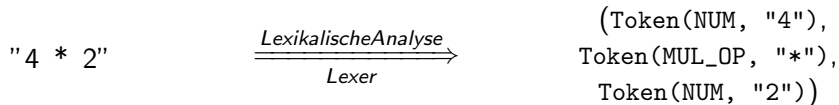


Abbildung 5: Aus Eingabewort Tokens generieren.

- ▶ im Appendix ab Folie 68 genauer erklärt.

Syntaktische Analyse

Aufgabe



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 71 genauer erklärt.

Syntaktische Analyse

Zwischenschritte

"4 * 2"

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

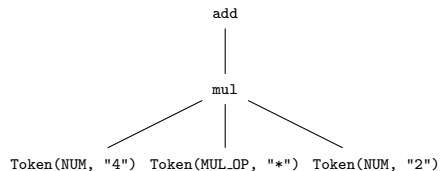
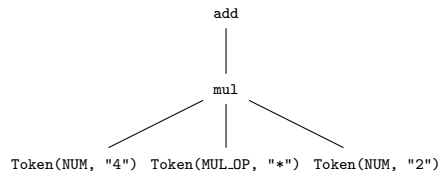


Abbildung 7: Aus dem Eingebewort einen Ableitungsbaum generieren.



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

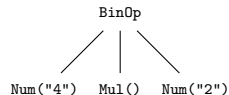


Abbildung 8: Aus Ableitungsbaum Abstrakten Syntaxbaum generieren.

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

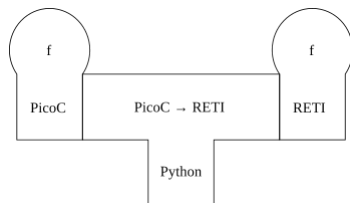


Abbildung 9: T-Diagramm für die Aufgabe der Code Generierung.

- ▶ 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.

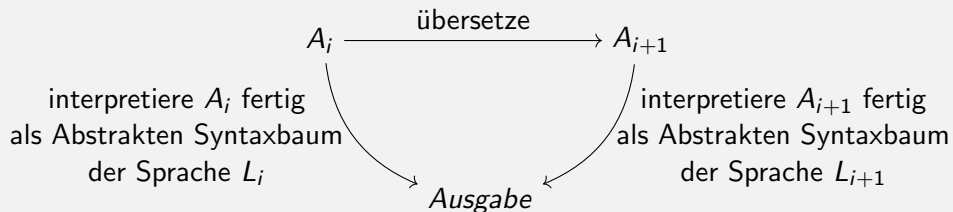
Code Generierung

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

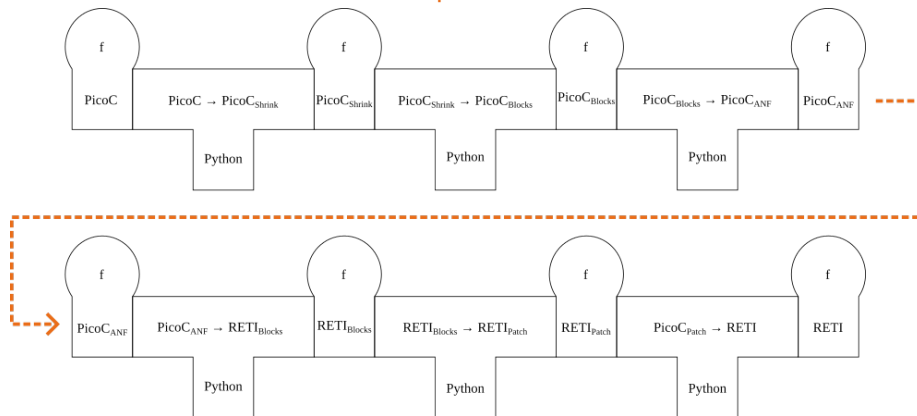
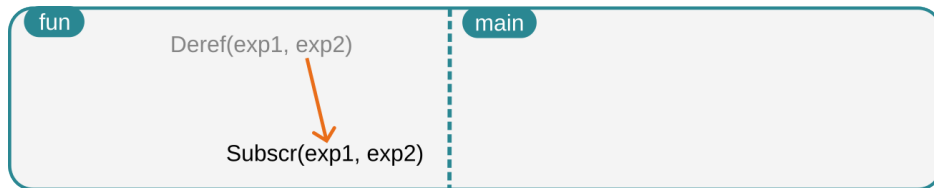


Abbildung 10: Architektur mit allen Passes ausgeschrieben.

Code Generierung

PicoC-Shrink Pass

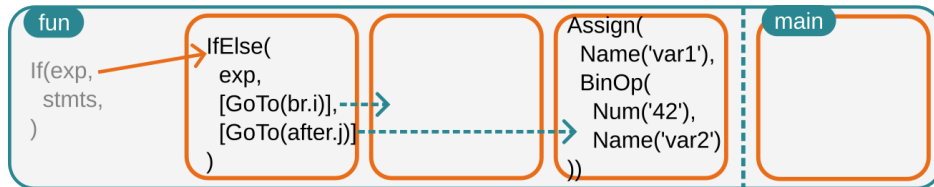
- ▶ gleiche Semantik des Dereferenzierungsoperators `*(pntr_or_ar + i)` und des Operators für Indexzugriff auf ein Feld `pntr_or_ar[i]`, sind austauschbar.
- ▶ Ersetzen von `Deref(exp, i)` durch `Subscr(exp, i)`.
- ▶ Dereferenzierung `*(pntr_or_ar + i)` wird von den Routinen für einen Indexzugriff auf 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 84 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 86 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 90 genauer erklärt.

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

Code

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

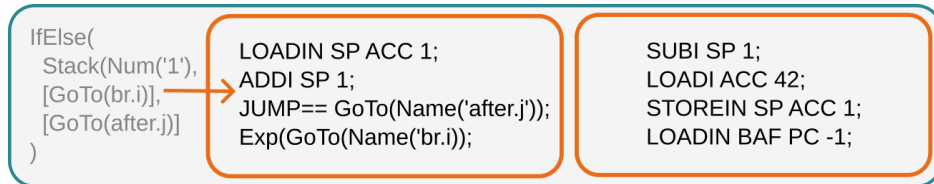
Code in A-Normalform

```
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 96 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 ab Folie 123 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 }

```

Datentyp der Variable

```

// ...
struct stt
↪ (*pntr_on_ar_of_sts)[3][2] =
↪ &ar_of_sts;
// ...

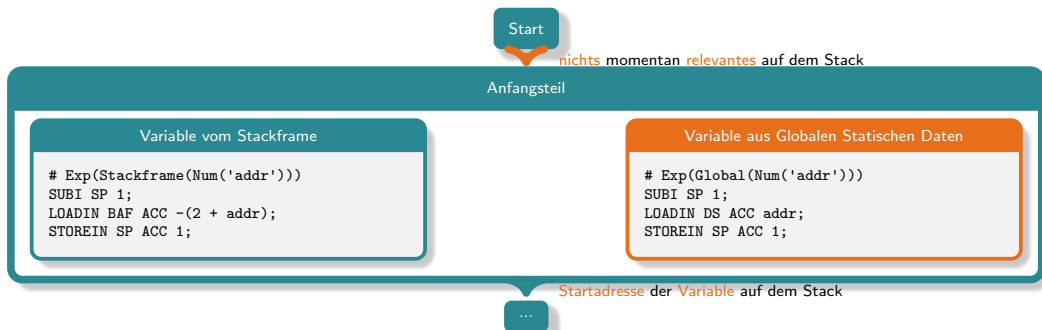
```

- ▶ ein „Zeiger auf ein Feld der Mächtigkeit 3 von Feldern der Mächtigkeit 2 von Verbunden des Typs stt“.
- ▶ *Clockwise/Spiral Rule*

Code Generierung

Codebeispiel

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



► $(*(\text{*ptr_on_ar_of_sts}+2))[1].\text{attr2}[1]$ struct stt $(\text{*ptr_on_ar_of_sts})[3][2] \dots$



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: 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

Schlussstil

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 101 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>
```

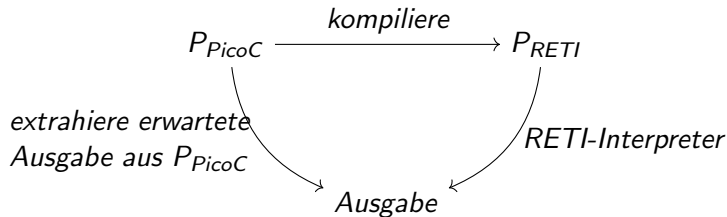
```
void main() {
    printf(" %d", 21 * 2);
    printf(" %d", 6 * 7);
}
```

- ▶ `// in:<space-sep-values>` sind **Eingaben** für die `input()`-Anweisungen.
- ▶ `// expected:<space-sep-values>` sind die **Erwarteten Ausgaben** der `print(exp)`-Anweisungen.
- ▶ `// datasegment:<size>` ist die **optionale Datensegmentgröße**.
- ▶ `convert_to_c.py`: **jedes** `print(exp)` wird durch `printf("%d", exp)` und **jedes** `input()` wird der Reihenfolge nach durch die **Eingaben** ersetzt.

Tests

Ablauf

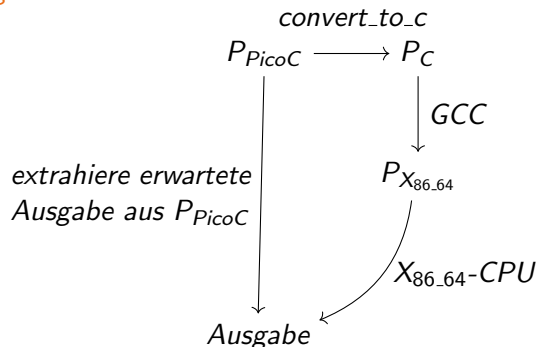
- ▶ prüfen, ob Erwartete Ausgaben und Ausgaben der `print(exp)`-Anweisungen identisch sind.
- ▶ Schreiber der Tests = Implementierer des PicoC-Compiler \Rightarrow Tests bestätigen nur das PicoC-Compiler genauso implementiert, wie diese Person die Semantik von L_{PicoC} interpretiert hat.



Tests

Ablauf

- ▶ der **GCC** setzt die **Semantik** von L_{PicoC} sehr wahrscheinlich korrekt um.
- ▶ prüfen, ob **Erwartete Ausgaben** und **Ausgaben der** `printf("%d", exp)-Anweisungen` identisch sind.



Tests

Durchlauf aller Tests

- `> make test <more-options>` (siehe Appendix auf Folie 118)

```
> 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

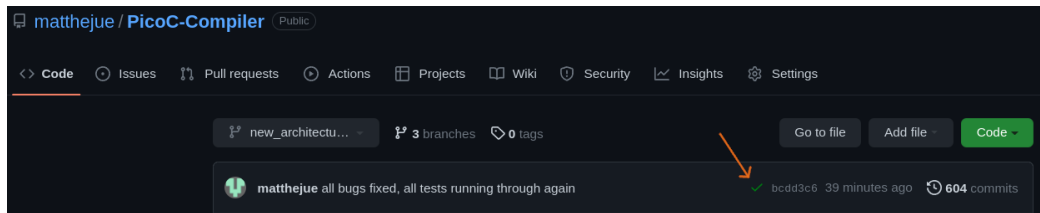


Abbildung 11: Autotesting mit GitHub Actions.

- ▶ <https://github.com/matthejue/PicoC-Compiler>.
- ▶ im Appendix auf Folie 119 sind die verschiedenen **Bedienmöglichkeiten** zum Ausführen der Tests erklärt.
- ▶ im Appendix auf Folie 122 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 105 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 111 aufgelistet.
- ▶ **Show-Mode:**
 - ▶ **Bedienung** des Show-Modes im Appendix ab Folie 114 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: 84
Instruction: STOREIN SP ACC 2;
ACC: 42
ACC_SIMPLE: 42
INI: 0
INI_SIMPLE: 0
IN2: 2
IN2_SIMPLE: 2
PC: 2147483709
PC_SIMPLE: 61
SP: 2147483786
SP_SIMPLE: 130
BAF: 2147483792
BAF_SIMPLE: 144
CS: 2147483651
CS_SIMPLE: 3
DS: 2147483766
DS_SIMPLE: 118
SRAM:
00000 JUMP 0;
00001 2147483648
00002 0
00003 CALL INPUT ACC; <- CS
00004 SUBI SP 1;
00005 STOREIN SP ACC 1;
00006 LOADIN SP ACC 1;
00007 STOREIN DS ACC 0;
00008 ADDI SP 1;
00009 SUBI SP 2;
00010 SUBI SP 1;
00011 LOADIN DS ACC 0;
00012 STOREIN SP ACC 1;
00013 MOVE BAF ACC;
00014 ADDI SP 3;
00015 MOVE SP BAF;
00016 SUBI SP 5;
00017 STOREIN BAF ACC 0;
00018 LOADI ACC 19;
00019 ADD ACC CS;
00020 STOREIN BAF ACC -1;
00021 JUMP 44;
00022 MOVE BAF INI;
00023 LOADIN INI BAF 0;
00024 MOVE INI SP;
00025 SUBI SP 1;
00026 STOREIN SP ACC 1;
00027 LOADIN SP ACC 1;
00028 STOREIN DS ACC 1;
00029 ADDI SP 1;
00030 SUBI SP 1;
00031 LOADIN DS ACC 1;
00032 STOREIN SP ACC 1;
00033 SUBI SP 1;
00034 LOADI ACC 2;
00035 STOREIN SP ACC 1;
00036 LOADIN SP ACC 2;
00037 LOADIN SP IN2 1;
00038 ADD ACC IN2;
00039 STOREIN SP ACC 2;
00040 ADDI SP 1;
00041 LOADIN SP ACC 1;
00042 ADDI SP 1;
00043 CALL PRINT ACC;
00044 LOADIN BAF PC -1;
00045 SUBI SP 1;
00046 LOADI ACC 2;
00047 STOREIN SP ACC 1;
00048 LOADIN SP ACC 1;
00049 STOREIN BAF ACC -3;
00050 ADDI SP 1;
00051 SUBI SP 1;
00052 LOADIN BAF ACC -2;
00053 STOREIN SP ACC 1;
00054 SUBI SP 1;
00055 LOADIN BAF -3;
00056 STOREIN SP ACC 1;
00057 LOADIN SP ACC 2;
00058 LOADIN SP IN2 1;
00059 ADDI SP 1;
00060 STOREIN SP ACC 2;
00061 CALL PRINT ACC;
00062 LOADIN SP ACC 1;
00063 STOREIN BAF ACC -4;
00064 ADDI SP 1;
00065 SUBI SP 1;
00066 LOADIN BAF ACC -4;
00067 STOREIN SP ACC 1;
00068 LOADIN SP ACC 1;
00069 ADDI SP 1;
00070 CALL PRINT ACC;
00071 SUBI SP 2;
00072 SUBI SP 1;
00073 LOADIN BAF ACC -4;
00074 STOREIN SP ACC 1;
00075 LOADIN SP ACC 1;
00076 ADDI SP 1;
00077 CALL PRINT ACC;
00078 SUBI SP 2;
00079 SUBI SP 1;
00080 LOADIN BAF ACC -4;
00081 STOREIN SP ACC 1;
00082 LOADIN SP ACC 1;
00083 ADDI SP 1;
00084 CALL PRINT ACC;
00085 SUBI SP 2;
00086 SUBI SP 1;
00087 LOADIN BAF ACC -4;
00088 STOREIN SP ACC 1;
00089 LOADIN SP ACC 1;
00090 ADDI SP 1;
00091 CALL PRINT ACC;
00092 SUBI SP 1;
00093 LOADIN BAF ACC -4;
00094 STOREIN SP ACC 1;
00095 MOVE BAF ACC;
00096 ADDI SP 3;
00097 MOVE SP BAF;
00098 SUBI SP 5;
00099 STOREIN BAF ACC 0;
00100 LOADI ACC 19;
00101 2147483678
00102 2147483658
00103 2147483797 <- BAF
00104 2147483792
00105 2147483786
00106 2147483780
00107 2147483774
00108 2147483768
00109 2147483762
00110 2147483756
00111 2147483750
00112 2147483744
00113 2147483738
00114 2147483732
00115 2147483726
00116 2147483720
00117 2147483714
00118 2147483708
00119 2147483702
00120 2147483696
00121 2147483690
00122 2147483684
00123 2147483678
00124 2147483672
00125 2147483666
00126 2147483660
00127 2147483654
00128 2147483648
00129 2147483642
00130 2147483636
00131 2147483630
00132 2147483624
00133 2147483618
00134 2147483612
00135 2147483606
00136 2147483600
00137 2147483594
00138 2147483588
00139 2147483582
00140 2147483576
00141 2147483570
00142 2147483564
00143 2147483558
00144 2147483552
00145 2147483546
00146 2147483540
00147 2147483534
00148 2147483528
00149 2147483522
00150 2147483516
00151 2147483510
00152 2147483504
00153 2147483498
00154 2147483492
00155 2147483486
00156 2147483480
00157 2147483474
00158 2147483468
00159 2147483462
00160 2147483456
00161 2147483450
00162 2147483444
00163 2147483438
00164 2147483432
00165 2147483426
00166 2147483420
00167 2147483414
00168 2147483408
00169 2147483402
00170 2147483396
00171 2147483390
00172 2147483384
00173 2147483378
00174 2147483372
00175 2147483366
00176 2147483360
00177 2147483354
00178 2147483348
00179 2147483342
00180 2147483336
00181 2147483330
00182 2147483324
00183 2147483318
00184 2147483312
00185 2147483306
00186 2147483300
00187 2147483294
00188 2147483288
00189 2147483282
00190 2147483276
00191 2147483270
00192 2147483264
00193 2147483258
00194 2147483252
00195 2147483246
00196 2147483240
00197 2147483234
00198 2147483228
00199 2147483222
00200 2147483216
00201 2147483210
00202 2147483204
00203 2147483198
00204 2147483192
00205 2147483186
00206 2147483180
00207 2147483174
00208 2147483168
00209 2147483162
00210 2147483156
00211 2147483150
00212 2147483144
00213 2147483138
00214 2147483132
00215 2147483126
00216 2147483120
00217 2147483114
00218 2147483108
00219 2147483102
00220 2147483096
00221 2147483090
00222 2147483084
00223 2147483078
00224 2147483072
00225 2147483066
00226 2147483060
00227 2147483054
00228 2147483048
00229 2147483042
00230 2147483036
00231 2147483030
00232 2147483024
00233 2147483018
00234 2147483012
00235 2147483006
00236 2147483000
00237 2147482994
00238 2147482988
00239 2147482982
00240 2147482976
00241 2147482970
00242 2147482964
00243 2147482958
00244 2147482952
00245 2147482946
00246 2147482940
00247 2147482934
00248 2147482928
00249 2147482922
00250 2147482916
00251 2147482910
00252 2147482904
00253 2147482898
00254 2147482892
00255 2147482886
00256 2147482880
00257 2147482874
00258 2147482868
00259 2147482862
00260 2147482856
00261 2147482850
00262 2147482844
00263 2147482838
00264 2147482832
00265 2147482826
00266 2147482820
00267 2147482814
00268 2147482808
00269 2147482802
00270 2147482796
00271 2147482790
00272 2147482784
00273 2147482778
00274 2147482772
00275 2147482766
00276 2147482760
00277 2147482754
00278 2147482748
00279 2147482742
00280 2147482736
00281 2147482730
00282 2147482724
00283 2147482718
00284 2147482712
00285 2147482706
00286 2147482700
00287 2147482694
00288 2147482688
00289 2147482682
00290 2147482676
00291 2147482670
00292 2147482664
00293 2147482658
00294 2147482652
00295 2147482646
00296 2147482640
00297 2147482634
00298 2147482628
00299 2147482622
00300 2147482616
00301 2147482610
00302 2147482604
00303 2147482598
00304 2147482592
00305 2147482586
00306 2147482580
00307 2147482574
00308 2147482568
00309 2147482562
00310 2147482556
00311 2147482550
00312 2147482544
00313 2147482538
00314 2147482532
00315 2147482526
00316 2147482520
00317 2147482514
00318 2147482508
00319 2147482502
00320 2147482496
00321 2147482490
00322 2147482484
00323 2147482478
00324 2147482472
00325 2147482466
00326 2147482460
00327 2147482454
00328 2147482448
00329 2147482442
00330 2147482436
00331 2147482430
00332 2147482424
00333 2147482418
00334 2147482412
00335 2147482406
00336 2147482400
00337 2147482394
00338 2147482388
00339 2147482382
00340 2147482376
00341 2147482370
00342 2147482364
00343 2147482358
00344 2147482352
00345 2147482346
00346 2147482340
00347 2147482334
00348 2147482328
00349 2147482322
00350 2147482316
00351 2147482310
00352 2147482304
00353 2147482298
00354 2147482292
00355 2147482286
00356 2147482280
00357 2147482274
00358 2147482268
00359 2147482262
00360 2147482256
00361 2147482250
00362 2147482244
00363 2147482238
00364 2147482232
00365 2147482226
00366 2147482220
00367 2147482214
00368 2147482208
00369 2147482202
00370 2147482196
00371 2147482190
00372 2147482184
00373 2147482178
00374 2147482172
00375 2147482166
00376 2147482160
00377 2147482154
00378 2147482148
00379 2147482142
00380 2147482136
00381 2147482130
00382 2147482124
00383 2147482118
00384 2147482112
00385 2147482106
00386 2147482100
00387 2147482094
00388 2147482088
00389 2147482082
00390 2147482076
00391 2147482070
00392 2147482064
00393 2147482058
00394 2147482052
00395 2147482046
00396 2147482040
00397 2147482034
00398 2147482028
00399 2147482022
00400 2147482016
00401 2147482010
00402 2147482004
00403 2147482000
00404 2147481994
00405 2147481988
00406 2147481982
00407 2147481976
00408 2147481970
00409 2147481964
00410 2147481958
00411 2147481952
00412 2147481946
00413 2147481940
00414 2147481934
00415 2147481928
00416 2147481922
00417 2147481916
00418 2147481910
00419 2147481904
00420 2147481898
00421 2147481892
00422 2147481886
00423 2147481880
00424 2147481874
00425 2147481868
00426 2147481862
00427 2147481856
00428 2147481850
00429 2147481844
00430 2147481838
00431 2147481832
00432 2147481826
00433 2147481820
00434 2147481814
00435 2147481808
00436 2147481802
00437 2147481796
00438 2147481790
00439 2147481784
00440 2147481778
00441 2147481772
00442 2147481766
00443 2147481760
00444 2147481754
00445 2147481748
00446 2147481742
00447 2147481736
00448 2147481730
00449 2147481724
00450 2147481718
00451 2147481712
00452 2147481706
00453 2147481700
00454 2147481694
00455 2147481688
00456 2147481682
00457 2147481676
00458 2147481670
00459 2147481664
00460 2147481658
00461 2147481652
00462 2147481646
00463 2147481640
00464 2147481634
00465 2147481628
00466 2147481622
00467 2147481616
00468 2147481610
00469 2147481604
00470 2147481598
00471 2147481592
00472 2147481586
00473 2147481580
00474 2147481574
00475 2147481568
00476 2147481562
00477 2147481556
00478 2147481550
00479 2147481544
00480 2147481538
00481 2147481532
00482 2147481526
00483 2147481520
00484 2147481514
00485 2147481508
00486 2147481502
00487 2147481496
00488 2147481490
00489 2147481484
00490 2147481478
00491 2147481472
00492 2147481466
00493 2147481460
00494 2147481454
00495 2147481448
00496 2147481442
00497 2147481436
00498 2147481430
00499 2147481424
00500 2147481418
00501 2147481412
00502 2147481406
00503 2147481400
00504 2147481394
00505 2147481388
00506 2147481382
00507 2147481376
00508 2147481370
00509 2147481364
00510 2147481358
00511 2147481352
00512 2147481346
00513 2147481340
00514 2147481334
00515 2147481328
00516 2147481322
00517 2147481316
00518 2147481310
00519 2147481304
00520 2147481298
00521 2147481292
00522 2147481286
00523 2147481280
00524 2147481274
00525 2147481268
00526 2147481262
00527 2147481256
00528 2147481250
00529 2147481244
00530 2147481238
00531 2147481232
00532 2147481226
00533 2147481220
00534 2147481214
00535 2147481208
00536 2147481202
00537 2147481196
00538 2147481190
00539 2147481184
00540 2147481178
00541 2147481172
00542 2147481166
00543 2147481160
00544 2147481154
00545 2147481148
00546 2147481142
00547 2147481136
00548 2147481130
00549 2147481124
00550 2147481118
00551 2147481112
00552 2147481106
00553 2147481100
00554 2147481094
00555 2147481088
00556 2147481082
00557 2147481076
00558 2147481070
00559 2147481064
00560 2147481058
00561 2147481052
00562 2147481046
00563 2147481040
00564 2147481034
00565 2147481028
00566 2147481022
00567 2147481016
00568 2147481010
00569 2147481004
00570 2147481000
00571 2147480994
00572 2147480988
00573 2147480982
00574 2147480976
00575 2147480970
00576 2147480964
00577 2147480958
00578 2147480952
00579 2147480946
00580 2147480940
00581 2147480934
00582 2147480928
00583 2147480922
00584 2147480916
00585 2147480910
00586 2147480904
00587 2147480898
00588 2147480892
00589 2147480886
00590 2147480880
00591 2147480874
00592 2147480868
00593 2147480862
00594 2147480856
00595 2147480850
00596 2147480844
00597 2147480838
00598 2147480832
00599 2147480826
00600 2147480820
00601 2147480814
00602 2147480808
00603 2147480802
00604 2147480796
00605 2147480790
00606 2147480784
00607 2147480778
00608 2147480772
00609 2147480766
00610 2147480760
00611 2147480754
00612 2147480748
00613 2147480742
00614 2147480736
00615 2147480730
00616 2147480724
00617 2147480718
00618 2147480712
00619 2147480706
00620 2147480700
00621 2147480694
00622 2147480688
00623 2147480682
00624 2147480676
00625 2147480670
00626 2147480664
00627 2147480658
00628 2147480652
00629 2147480646
00630 2147480640
00631 2147480634
00632 2147480628
00633 2147480622
00634 2147480616
00635 2147480610
00636 2147480604
00637 2147480598
00638 2147480592
00639 2147480586
00640 2147480580
00641 2147480574
00642 2147480568
00643 2147480562
00644 2147480556
00645 2147480550
00646 2147480544
00647 2147480538
00648 2147480532
00649 2147480526
00650 2147480520
00651 2147480514
00652 2147480508
00653 2147480502
00654 2147480496
00655 2147480490
00656 2147480484
00657 2147480478
00658 2147480472
00659 2147480466
00660 2147480460
00661 2147480454
00662 2147480448
00663 2147480442
00664 2147480436
00665 2147480430
00666 2147480424
00667 2147480418
00668 2147480412
00669 2147480406
00670 2147480400
00671 2147480394
00672 2147480388
00673 2147480382
00674 2147480376
00675 2147480370
00676 2147480364
00677 2147480358
00678 2147480352
00679 2147480346
00680 2147480340
00681 2147480334
00682 2147480328
00683 2147480322
00684 2147480316
00685 2147480310
00686 2147480304
00687 2147480298
00688 2147480292
00689 2147480286
00690 2147480280
00691 2147480274
00692 2147480268
00693 2147480262
00694 2147480256
00695 2147480250
00696 2147480244
00697 2147480238
00698 2147480232
00699 2147480226
00700 2147480220
00701 2147480214
00702 2147480208
00703 2147480202
00704 2147480196
00705 2147480190
00706 2147480184
00707 2147480178
00708 2147480172
00709 2147480166
00710 2147480160
00711 2147480154
00712 2147480148
00713 2147480142
00714 2147480136
00715 2147480130
00716 2147480124
00717 2147480118
00718 2147480112
00719 2147480106
00720 2147480100
00721 2147480094
00722 2147480088
00723 2147480082
00724 2147480076
00725 2147480070
00726 2147480064
00727 2147480058
00728 2147480052
00729 2147480046
00730 2147480040
00731 2147480034
00732 2147480028
```


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, 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 und Dokumentation

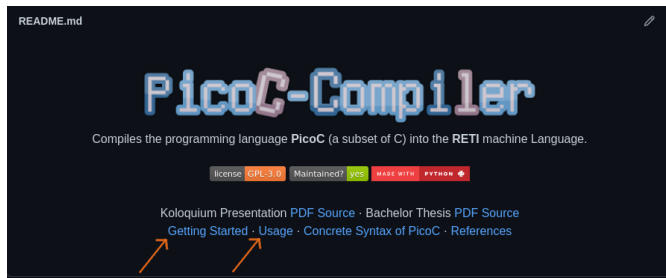


Abbildung 13: README.md des PicoC-Compilers Repositories.

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

Appendix

Schwerpunkte

- ▶ **Syntax** und **Semantik** der Sprache L_{PicoC} **identisch** zur Sprache L_C .
 - ▶ außer bei Kommandozeilenoptionen, Fehlermeldungen usw. **kein Unterschied** zu z.B. dem **GCC**.
- ▶ möglichst die **RETI-Codeschnipsel** aus der Vorlesung Scholl, „Betriebssysteme“, Kapitel 3 Übersetzung höherer Programmiersprachen in Maschinensprache.
 - ▶ bei **Inkonsistenzen** und **Umstimmigkeiten** angepasst.

PicoC

Grammatik

- ▶ https://github.com/matthejue/PicoC-Compiler/blob/new_architecture/src/concrete_syntax_picoc.lark

RETI-Architektur

Grammatik

dig_no_0	::=	"1" "2" "3" "4" "5" "6"	<i>L_Program</i>
		"7" "8" "9"	
dig_with_0	::=	"0" <i>dig_no_0</i>	
num	::=	"0" <i>dig_no_0</i> <i>dig_with_0</i> * "-" <i>dig_no_0</i> *	
letter	::=	"a" ... "Z"	
name	::=	<i>letter</i> (<i>letter</i> <i>dig_with_0</i> _)*	
reg	::=	"ACC" "IN1" "IN2" "PC" "SP"	
		"BAF" "CS" "DS"	
arg	::=	<i>reg</i> <i>num</i>	
rel	::=	"==" "!=" "<" "<=" ">"	
		">=" "_NOP"	

Grammatik 2: Grammatik des Lexers für die Sprache L_{RETI} in EBNF.

RETI-Architektur

Grammatik

instr	::=	"ADD" reg arg "ADDI" reg num "SUB" reg arg	<i>L_Program</i>
		"SUBI" reg num "MULT" reg arg "MULTI" reg num	
		"DIV" reg arg "DIVI" reg num "MOD" reg arg	
		"MODI" reg num "OPLUS" reg arg "OPLUSI" reg num	
		"OR" reg arg "ORI" reg num	
		"AND" reg arg "ANDI" reg num	
		"LOAD" reg num "LOADIN" arg arg num	
		"LOADI" reg num	
		"STORE" reg num "STOREIN" arg argnum	
		"MOVE" reg reg	
		"JUMP" rel num INT num RTI	
		"CALL" "INPUT" reg "CALL" "PRINT" reg	
program	::=	(instr";")*	

Grammatik 3: Grammatik des Parsers für die Sprache L_{RETI} in EBNF.

PicoC

Definitionen

Call-by-Value

- ▶ **Kopie** des **Arguments** wird im Stackframe der aufgerufenen Funktion an **Parameter** gebunden.
- ▶ **Argument** bleibt bei **Änderungen** am entsprechenden **Parameter** in der aufgerufenen Funktion in der aufrufenden Funktion **unverändert**.

Call-by-Reference

- ▶ **Referenz** des **Arguments** wird im Stackframe der aufgerufenen Funktion an **Parameter** gebunden.
- ▶ **Argument** ändert sich bei **Änderungen** am entsprechenden **Parameter** in der aufgerufenen Funktion auch in der aufrufenden Funktion .

Weitere Definitionen

Interpreter

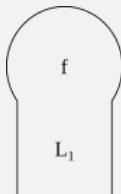


- ▶ führt Anweisungen „direkt“ aus.

T-Diagramm Programm



- ▶ in der Sprache L_1 geschrieben und berechnet Funktion f .

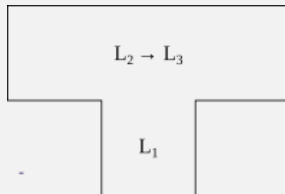


Weitere Definitionen

T-Diagramm Übersetzer



- ▶ in der Sprache L_1 geschrieben und übersetzt von Sprache L_2 in die Sprache L_3 .
- ▶ gleiche Semantik.
- ▶ Kompilieren ist immer Übersetzen, aber Übersetzen ist nicht immer Kompilieren.



Lexikalische Analyse

Aufgabe

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

Grammatik 4: Grammatik des Lexers

"4 * 2"	$\xrightarrow[\text{Lexer}]{\text{LexikalischeAnalyse}}$	(Token(NUM, "4"), Token(MUL_OP, "*"), Token(NUM, "2"))
---------	--	--

Abbildung 14: Aus Eingabewort Tokens generieren.

Lexikalische Analyse

Definitionen

Token



- ▶ Tupel (T, V) , wobei:
 - ▶ Tokentyp $T \hat{=}$
 - ▶ bestimmtes Nicht-Terminalsymbol auf der linken Seite des $::=$ -Symbols in der Grammatik des Lexers.
 - ▶ Überbegriff für möglicherweise unendliche Menge von Tokenwerten, die sich aus einem bestimmten Nicht-Terminalsymbol ableiten lassen.
 - ▶ in der Grammatik des Parsers ein Terminalsymbol.
 - ▶ Tokenwert $V \hat{=}$ aus einem bestimmten Nicht-Terminalsymbol ableitbares Wort in der Grammatik des Lexers.

Lexikalische Analyse

Definitionen

NUM	::=	"4" "2"	L_Lex
ADD_OP	::=	"+"	
MUL_OP	::=	"*"	

Grammatik 5: Grammatik des Lexers in EBNF

Lexer

- ▶ bildet **Eingabewort** $w \in \Sigma^*$ auf **Folge von Tokens** $(t_1, v_n) \dots (t_n, v_n) \in (T \times V)^*$ ab: $lex : \Sigma^* \rightarrow (T \times V)^*, w \mapsto (t_1, v_1) \dots (t_n, v_n)$.

Syntaktische Analyse

Ausgelassene Zwischenschritte

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

Grammatik 6: Grammatik des Parsers unten und Grammatik des Lexers oben

► Tokentypen *T* sind in der Grammatik des Parsers **Terminalsymbole**

add \Rightarrow *mul* \Rightarrow *mul MUL_OP NUM* \Rightarrow *NUM MUL_OP NUM* \Rightarrow * "4" "*" "2"

Syntaktische Analyse

Ausgelassene Zwischenschritte

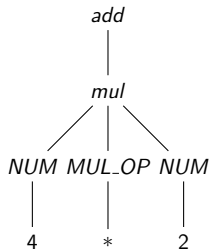


Abbildung 15: Formaler Ableitungsbaum

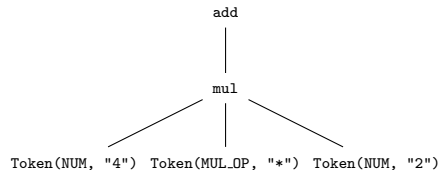


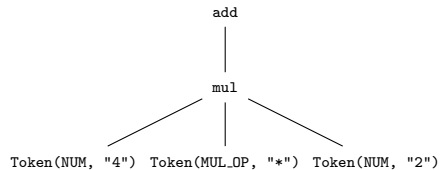
Abbildung 16: Compilerinterner Ableitungsbaum

Syntaktische Analyse

Ausgelassene Zwischenschritte

"4 * 2"

Parser
→
Lexer



- **Lexer** ist Teil des **Parsers**.

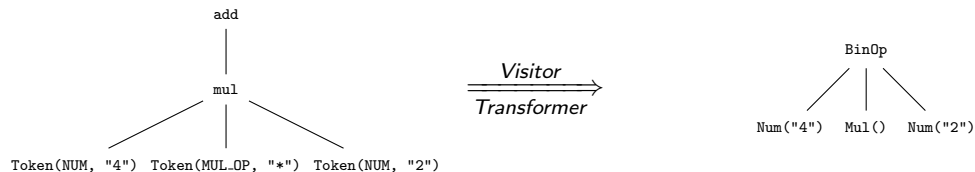
Syntaktische Analyse

Ausgelassene Zwischenschritte

```

bin_op ::= Add() | Mul()
exp    ::= BinOp(<exp>, <bin_op>, <exp>) | Num(<str>)
  
```

Grammatik 7: Produktionen für Abstrakten Syntaxbaum



Syntaktische Analyse

Lark Parsing Toolkit

- ▶ erleichtert Syntaktische Analyse.
- ▶ Grammatik spezifizieren nach der Lark ein Eingabewort parst und einen Ableitungsbaum generiert.
- ▶ Earley Parser implementiert.
- ▶ Implementierung von Visitor und Transformer.
- ▶ Quellcode: <https://github.com/lark-parser/lark>.
- ▶ Dokumentation:
<https://lark-parser.readthedocs.io/en/latest/index.html>.

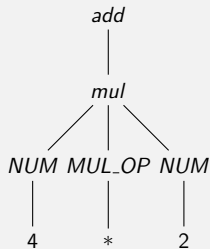
Syntaktische Analyse

Definitionen

Formaler Ableitungsbaum



- ▶ Darstellung einer **Ableitung** als **Baum**.
- ▶ Innere Knoten $\hat{=}$ Nicht-Terminalsymbole.
- ▶ Blätter $\hat{=}$ Terminalsymbole oder das **leere Wort** ε .



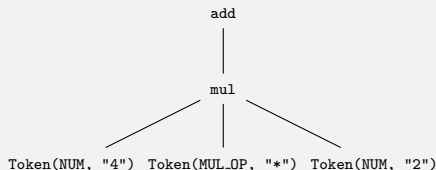
Syntaktische Analyse

Definitionen

(Compilerinterner) Ableitungsbaum



- ▶ compilerinterne Datenstruktur für Formalen Ableitungsbaum
- ▶ Innere Knoten $\hat{=}$ Nicht-Terminalsymbolen N der Grammatik des Parsers
 $G = \langle N, \Sigma, P, S \rangle$
- ▶ Blätter $\hat{=}$ Tokens (T, V) , Grammatik des Lexers interessiert nicht mehr



Syntaktische Analyse

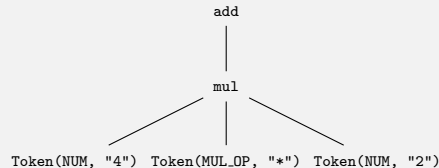
Definitionen

Parser

- ▶ generiert aus Eingabewort einen compilerinternen Ableitungsbaum
- ▶ beinhaltet Lexer

"4 * 2"

⇒



Syntaktische Analyse

Definitionen

Visitor



- ▶ von **unten-nach-oben** nach Prinzip der **Breitensuche** über **Ableitungsbaum**.
- ▶ **manipuliert** Knoten oder **tauscht** Knoten **in-place** mit anderen Knoten des Ableitungsbaumes, indem beim Antreffen bestimmter Knoten des Ableitungsbaumes bestimmte **Aktionen** ausgeführt werden.

Transformer



- ▶ von **unten-nach-oben** nach Prinzip der **Breitensuche** über **Ableitungsbaum**.
- ▶ generiert **Abstrakten Syntaxbaum**, indem beim Antreffen bestimmter Knoten des Ableitungsbaumes, diese durch Knoten des Abstrakten Syntaxbaumes **ersetzt** werden.

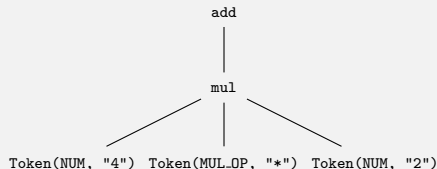
Syntaktische Analyse

Definitionen

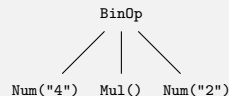
Abstrakter Syntaxbaum



- ▶ **compilerinterne** Datenstruktur
- ▶ **Abstraktion** eines Ableitungsbaumes, Knoten für z.B. Präzedenz sind weg.
- ▶ leichter **Zugriff** und **Weiterverarbeitbarkeit**
- ▶ setzt **Funktionalität einer Sprache** um und erlaubt es schnell herauszufinden aus welchen **Bestandteilen** der Sprache mit **unterscheidbarer Semantik** diese zusammengesetzt ist.



⇒



Code Generierung

Definitionen

Konkrete Syntax

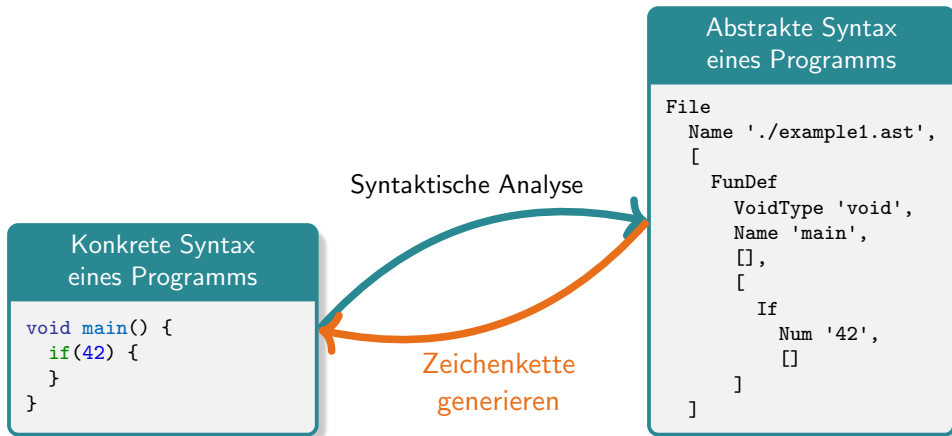
- ▶ bezeichnet den **Aufbau** von Programmen, wie man sie in eine **Textdatei** schreibt, um sie **kompilieren** zu lassen.

Abstrakte Syntax

- ▶ bezeichnet den **Aufbau** von **Abstrakten Syntaxbäumen**.
- ▶ nur bestimmte **Kompositionen** von Knoten sind **erlaubt**.

Code Generierung

Definitionen



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)` mithilfe von `Block(name, stmts_instrs-`, `GoTo(label)-` und `IfElse(exp, stmts1, stmts2)` umgesetzt.
 - ▶ für **Bedingungen** und **Branches** ist jeweils ein eigener **Block** zuständig.
 - ▶ `IfElse(exp, stmts1, stmts2)` wird zur Umsetzung von **Bedingungen** verwendet.
 - ▶ für beide Fälle, wenn die Bedingung **wahr** oder **falsch** ist, wird mithilfe von `GoTo(label)` in einen von zwei **alternativen Blöcken** gesprungen oder ein Block **erneut aufgerufen** usw.

Code Generierung

PicoC-Blocks Pass

- ▶ jede **Funktion** erhält **eigenen Block**, der alle Anweisungen bis zum ersten Auftauchen oder Nicht-Auftauchen eines `If(exp, stmts)`, `While(exp, stmts)` oder `DoWhile(exp, stmts)` enthält.

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** mit ihren **Lokalen Variablen**, **Parametern** und **Sichtbarkeitsbereichen**, sowie **Verbunstypen** mit ihren **Verbundsattributen** werden mithilfe einer **Symboltabelle** aufgelöst.

Symboltabelle



- ▶ $sym : \{my_var, my_fun, \dots\} \rightarrow \{Adresse, Datentyp, \dots\}^n, Bezeichner \mapsto (Information_1, \dots, Information_n).$
- ▶ um **während** dem **Kompilervorang** Informationen zu speichern, die später **nicht** mehr so einfach **zugänglich** sind.

Code Generierung

PicoC-ANF Pass

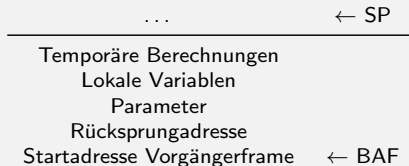
- ▶ alle Funktionen, **außer** der `main`-Funktion besitzen einen **Stackframe** für **Lokale Variablen** und **Parameter**.
- ▶ **Globale Statische Daten** sind **Globale Variablen**, sowie **Lokale Variablen** und **Parameter** der `main`-Funktion.

Code Generierung

PicoC-ANF Pass

Stackframe

- ▶ Datenstruktur, um **Zustand** einer Funktion zur Laufzeit zu „konservieren“.
- ▶ in einem Stack **übereinander gestapelt** und in die **entgegengesetzte Richtung** wieder abgebaut.
- ▶ die **Startadresse** des **Vorgängerframes** und die **Rücksprungadresse** beide im Stackframe der **aufgerufenen** Funktion.



Code Generierung

PicoC-ANF Pass

- **Zweck:** Maschinenbefehlen annähern, die meist nur **eine Aktion** ausführen, indem eine **Anweisung**, die **mehreren Aktionen** entspricht, aufgespalten wird und **Nebeneffekte** vorgeschoben werden.

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);  
}
```

PicoC-ANF Pass

A-Normalform

Unreiner Ausdruck



- ▶ Ausdruck mit Nebeneffekt.

Reiner Ausdruck



- ▶ Ausdruck ohne Nebeneffekt.

PicoC-ANF Pass

A-Normalform

Monadische Normalform

- ▶ alle Anweisungen enthalten keine Unreinen Ausdrücke.
- ▶ Reine und Unreine Ausdrücke voneinander getrennt.

Code

```
void main() {  
    int var = 5 % 4;  
}
```

ziehe Unreine Ausdrücke
aus Anweisungen vor

Code in Monadi- scher Normalform

```
void main() {  
    int var;  
    var = 5 % 4;  
}
```

PicoC-ANF Pass

A-Normalform

Atomarer Ausdruck



- ▶ übersetzt sich in **keinen** kompletten Maschinenbefehl und **keine** Folge von Maschinenbefehlen.
- ▶ **legt** z.B. einen **Immediate** in einem Maschinenbefehl **fest**.

Komplexer Ausdruck



- ▶ ein Ausdruck, der **nicht atomar** ist.
- ▶ lässt sich in einen **Maschinenbefehl** oder eine **Folge von Maschinenbefehlen** übersetzen.

PicoC-ANF Pass

A-Normalform

A-Normalform

- ▶ ist bereits in **Monadischer Normalform**.
- ▶ alle **Anweisungen** und **Ausdrücke** enthalten ausschließlich **Atomare Ausdrücke**.

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

Code

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

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);  
}
```

PicoC-ANF Pass

A-Normalform

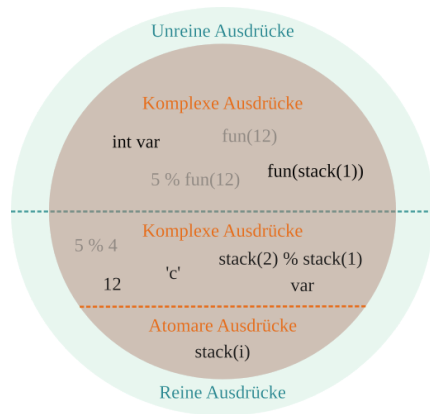


Abbildung 17: Überblick über Komplexe, Atomare, Unreine und Reine Ausdrücke.

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.
 - ▶ RETI-Code **vor** jede Division, der prüft, ob durch 0 geteilt wird.
 - ▶ Fehlercode 1 in ACC-Register für `DivisionByZero`.
 - ▶ Ausführung wird **beendet**.

Code Generierung

RETI-Patch Pass

- ▶ **Ausbessern** (engl. to patch) des **Abstrakten Syntaxbaumes** durch:
 - ▶ Überprüfen, ob **Immediates** $\text{Im}(\text{str})$ in Befehlen $< -(2^{21})$ oder $> 2^{21} - 1$.
 - ▶ **Bitshiften** und Anwenden von **Bitweise ODER**.
 - ▶ **Immediate** $< -(2^{31})$ oder $> 2^{31} - 1 \Rightarrow \text{TooLargeLiteral}$.

Code Generierung

RETI Pass

- ▶ letzte verbliebene **PicoC-Knoten** werden durch entsprechende **RETI-Knoten** ersetzt:
 - ▶ **keine Blöcke** mehr, Knoten genauso **zusammengefügt**, wie sie in entfernten Blöcken **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.

Code Generierung

RETI Pass

▶ $adr_{danach} = \#Bef_{vor\ akt.\ Bl.} + idx + 4$

▶ $Dist_{Zielbl.} = \begin{cases} \#Bef_{vor\ Zielbl.} - \#Bef_{vor\ akt.\ Bl.} - idx & \#Bef_{vor\ Zielbl.} \neq \#Bef_{vor\ akt.\ Bl.} \\ -idx & \#Bef_{vor\ Zielbl.} = \#Bef_{vor\ akt.\ Bl.} \end{cases}$

Code Generierung

RETI Pass

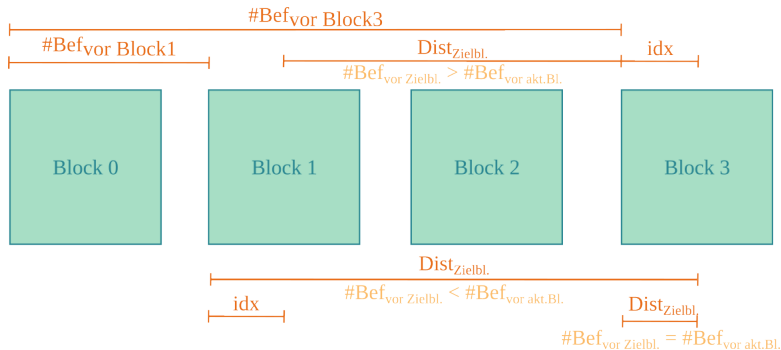


Abbildung 18: Veranschaulichung Distanzberechnung.

Fehlermeldungen

Kategorien

Fehlerkategorie	Beschreibung
UnexpectedCharacter	Der Lexer ist auf eine unerwartete Zeichenfolge gestossen, die in der Grammatik des Lexers nicht abgeleitet werden kann.
UnexpectedToken	Der Parser hat ein unerwartetes Token erhalten, das in dem Kontext in dem es sich befand in der Grammatik des Parsers nicht abgeleitet werden kann.
UnexpectedEOF	Der Parser hat in dem Kontext in dem er sich befand bestimmte Tokens erwartet , aber die Eingabe endete abrupt.

Tabelle 1: Fehlerarten in der Lexikalischen und Syntaktischen Analyse.

Fehlerkategorie	Beschreibung
DivisionByZero	Wenn bei einer Division durch 0 geteilt wird (z.B. <code>var / 0</code>).

Tabelle 2: Fehlerarten, die zur Laufzeit auftreten.

Fehlermeldungen

Kategorien, Teil 2

Fehlerkategorie	Beschreibung
UnknownIdentifier	Es wird ein Zugriff auf einen Bezeichner gemacht (z.B. <code>unknown_var + 1</code>), der noch nicht deklariert und ist daher nicht in der Symboltabelle aufgefunden werden kann.
UnknownAttribute	Der Verbundstyp (z.B. <code>struct st {int attr1; int attr2;}</code>) auf dessen Attribut im momentanen Kontext zugegriffen wird (z.B. <code>var[3].unknown_attr</code>) besitzt das Attribut (z.B. <code>unknown_attr</code>) auf das zugegriffen werden soll nicht .
ReDeclarationOrDefinition	Ein Bezeichner von z.B. einer Funktion oder Variable , der bereits deklariert oder definiert ist (z.B. <code>int var</code>) wird erneut deklariert oder definiert (z.B. <code>int var[2]</code>). Dieser Fehler ist leicht festzustellen, indem geprüft wird ob das Assoziative Feld durch welches die Symboltabelle umgesetzt ist diesen Bezeichner bereits als Schlüssel besitzt.
TooLargeLiteral	Der Wert eines Literals ist größer als $2^{31} - 1$ oder kleiner als -2^{31} .
NoMainFunction	Das Programm besitzt keine oder mehr als eine main-Funktion.

Tabelle 3: Fehlerarten in den Passes.

Fehlermeldungen

Kategorien, Teil 3

Fehlerkategorie	Beschreibung
ConstAssign	Wenn einer initialisierten Konstante (z.B. <code>const int const_var = 42</code>) ein Wert zugewiesen wird (z.B. <code>const_var = 41</code>). Der einzigste Weg , wie eine Konstante einen Wert erhält ist bei ihrer Initialisierung .
PrototypeMismatch	Der Prototyp einer deklarierten Funktion (z.B. <code>int fun(int arg1, int arg2[3])</code>) stimmt nicht mit dem Prototyp der späteren Definition dieser Funktion (z.B. <code>void fun(int arg1[2], int arg2) { }</code>) überein.
ArgumentMismatch	Wenn die Argumente eines Funktionsaufrufs (z.B. <code>fun(42, 314)</code>) nicht mit dem Prototyp der Funktion die aufgerufen werden soll (z.B. <code>void fun(int arg[2]) { }</code>) nach Datentypen oder Anzahl Argumente bzw. Parameter übereinstimmt.
WrongReturnType	Wenn eine Funktion, die ihrem Prototyp zufolge einen Rückgabewert hat, der nicht mit dem dem Datentyp übereinstimmt, der von einer return-Anweisung zurückgegeben wird.

Tabelle 4: Fehlerarten in den Passes, Teil 2.

Fehlermeldungen

Kategorien, Teil 4

Fehlerkategorie	Beschreibung
DatatypeMismatch	Wenn die Operation und der Datentyp des Attributes oder Elementes auf welches in diesem Kontext zugegriffen wird nicht zueinander passen .

Tabelle 5: Fehlerarten in den Passes, Teil 3.

Bedienung des PicoC-Compilers

Kommandozeilenargumente <cli-options> für den Compiler

Kommandozeilen-option	Beschreibung	Standardwert
<code>-i, --intermediate_stages</code>	Gibt Zwischenstufen der Kompilierung in Form der verschiedenen Tokens , Ableitungsbäume , Abstrakten Syntaxbäume der verschiedenen Passes in Dateien mit entsprechenden Dateieindungen aber gleichem Basisnamen aus. Wenn die <code>--run</code> -Option aktiviert ist, wird der Zustand der RETI nach der Ausführung des letzten Befehls in eine Datei ausgegeben. Im Shell-Mode erfolgt keine Ausgabe in Dateien, sondern nur im Terminal .	False , most_used: True
<code>-p, --print</code>	Gibt alle Dateiausgaben auch im Terminal aus. Diese Option ist im Shell-Mode dauerhaft aktiviert.	False , Shell-Mode und most_used: True

Bedienung des PicoC-Compilers

Kommandozeilenargumente <cli-options> für den Compiler, Teil 2

Kommandozeilen-option	Beschreibung	Standardwert
-v, --verbose	Fügt den verschiedenen Zwischenschritten der Kompilierung , unter anderem auch dem finalen RETI-Code Kommentare hinzu. Diese Kommentare beinhalten eine Anweisung oder einen Befehl aus einem vorherigen Pass , der durch die darunterliegenden Anweisungen oder Befehle ersetzt wurde. Wenn die --run und die --immediate_stages-Option aktiviert sind, wird der Zustand der virtuellen RETI-CPU vor und nach jedem Befehl ausgegeben.	False
-vv, --double_verbose	Hat dieselben Effekte , wie die --verbose-Option, aber bewirkt zusätzlich weitere Effekte . PicoC-Knoten erhalten bei der Ausgabe als zusammenhängende Abstrakte Syntaxbäume zusätzliche runde Klammern , sodass direkter abgelesen werden kann, wo ein Knoten anfängt und wo einer aufhört. In Fehlermeldungen werden mehr Tokens angezeigt, die an der Stelle der Fehlermeldung erwartet worden wären. Bei Aktivierung der --intermediate_stages-Option werden in den dadurch ausgegebenen Abstrakten Syntaxbäumen zusätzlich versteckte Attribute angezeigt, die Informationen zu Datentypen und Informationen für Fehlermeldungen beinhalten.	False

Bedienung des PicoC-Compilers

Kommandozeilenargumente <cli-options> für den Compiler, Teil 3

Kommandozeilen-option	Beschreibung	Standardwert
-h, --help	Zeigt diese Dokumentation mithilfe des im Betriebssystem eingestellten PDF-Viewers an.	False
-l, --lines	Es lässt sich einstellen, wieviele Zeilen rund um die Stelle an welcher ein Fehler aufgetreten ist angezeigt werden sollen.	2
-c, --color	Aktiviert farbige Ausgabe für Fehlermeldungen , PicoC- und RETI-Code , Tokens , Ableitungsbäume und Abstrakte Syntaxbäume der verschiedenen Passes.	False, most_used: True
-e, --example	Filtert für übersichtliche Codebeispiele bestimmte Kommentare in den Abstrakten Syntaxbäumen heraus. Diese Option wurde für die Codebeispiele in der schriftlichen Ausarbeitung der Bachelorarbeit implementiert.	False

Bedienung des PicoC-Compilers

Kommandozeilenargumente <cli-options> für den Compiler, Teil 4

Kommandozeilen-option	Beschreibung	Standardwert
-t, --traceback	Nutzt das Python Package traceback um bei Fehlermeldungen Stacktraces des Compilers auszugeben.	False
-d, --debug	Startet den PuDB-Debugger (pip install pudb) vor Beginn des Kompilierens oder Interpretierens.	False
-s, --supress_errors	Obwohl eine Fehlermeldung ausgegeben werden müsste, wird bei manchen Fehlermeldungen die Ausgabe unterdrückt .	False

Bedienung des PicoC-Compilers

Kommandozeilenargumente <cli-options> für den Interpreter

Kommandozeilen-option	Beschreibung	Standardwert
-R, --run	Führt die RETI-Befehle , die das Ergebnis der Kompilierung sind mit einer virtuellen RETI-CPU aus. Wenn die --intermediate_stages -Option aktiviert ist, wird eine Datei <basename>.reti_states erstellt, welche den Zustand der RETI-CPU nach dem letzten ausgeführten RETI-Befehl enthält. Wenn die --verbose - oder --double_verbose -Option aktiviert ist, wird der Zustand der RETI-CPU vor und nach jedem Befehl auch noch zusätzlich in die Datei <basename>.reti_states ausgegeben.	False , Show-Mode und most_used : True
-B, --process_begin	Setzt die Adresse , wo der Prozess bzw. das Codesegment für das ausgeführte Programm beginnt .	3
-D, --datasegment_size	Setzt die Größe des Datensegments . Diese Option muss mit Vorsicht gesetzt werden, denn wenn der Wert zu niedrig gesetzt wird, dann können die Globalen Statischen Daten und der Stack miteinander kollidieren .	32

Bedienung des PicoC-Compilers

Kommandozeilenargumente <cli-options> für den Interpreter, Teil 2

Kommandozeilenoption	Beschreibung	Standardwert
-S, --show_mode	Startet den Show-Mode . Der Show-Mode zeigt eine Zeichenfolge über mehrere Seiten verteilt an. Standardmäßig wird dies für die Zustände der RETI nach und vor der Ausführung eines bestimmten RETI-Befehls gemacht. Der Eindruck des Debuggens kommt dadurch, dass durch Drücken entsprechender Tasten immer an die richtigen Stellen gesprungen wird, an denen der nächste oder vorherige Zustand anfähgt .	False , compile_show und interpret_show: True
-P, --pages	Setzt auf wieviele Seiten im Show-Mode eine Zeichenfolge verteilt werden soll.	5
-E, --extension	Setzt welcher Dateityp , der durch eine bestimmte Dateiendung spezifiziert ist im Show-Mode angezeigt werden soll.	reti_states

Bedienung des PicoC-Compilers

Shell-Mode

- ▶ Starten: `> picoc_compiler .`
- ▶ Kompilieren: `> compile <cli-options> "<seq-of-stmts>" (cpl)`.
 - ▶ automatisch in main-Funktion eingefügt: `void main(){<seq-of-stmts>}`.
- ▶ Kompilieren, meist genutzt:
`> must_used <cli-options> "<seq-of-stmts>" (mu)`.
- ▶ Kompilieren und dann Show-Mode:
`> compile-show <cli-options> "<seq-of-stmts>" (cs)`.
- ▶ Interpretieren und dann Show-Mode:
`> interpret-show <cli-options> "<seq-of-instrs>" (is)`.

Bedienung des PicoC-Compilers

Shell-Mode, Teil 2

- ▶ Beenden: `> quit`.
- ▶ Dokumentation: `> help` (`(?)`).
- ▶ Multiline-Command: weitere Zeile mit `↵` und mit `;` terminieren.
- ▶ Farben toggeln: `> color_toggle` (`(ct)`).
- ▶ Cursor bewegen: `←`, `→`.
- ▶ Befehlshistorie: `↑`, `↓`.
- ▶ Autovervollständigung: `Tab`.

Bedienung des PicoC-Compilers

Shell-Mode, Teil 3

- ▶ Befehlshistorie anzeigen: `> history` .
- ▶ Aktion mit Befehlshistorie ausführen `> history <opt>` .
 - ▶ Befehl erneut ausführen: `-r <cmd-nr>` .
 - ▶ Befehl editieren `-e <cmd-nr>` (Editor durch **Environment Variable** `$EDITOR` bestimmt).
 - ▶ Befehlshistorie leeren: `-c` .
 - ▶ Befehl suchen: `ctrl + r` .

Bedienung des PicoC-Compilers

Show-Mode

- ▶ Starten: `> picoc_compiler -S <cli-options> program.(reti|picoc)` .
- ▶ Shell-Mode Befehle: `> cs <cli-options> "<seq-of-stmts>"` bzw.
`> is <cli-options> "<seq-of-instrs>"` .
- ▶ Anzahl Seiten: `-P <num>` .
- ▶ Dateiendung der gewünschten Datei: `-E <extension>` .
- ▶ Spezielle Einstellungen: `/interp_showcase.vim` .
- ▶ Neovim: `:help` , `:Tutor` .

Bedienung des PicoC-Compilers

Show-Mode, Teil 2

- ▶ Zustände vor / nach Befehl ansehen: **Tab**, **↑ -Tab**.
- ▶ Beenden: **q**, **Esc**.
- ▶ Fenster minimieren / maximieren: **m**, **M**.
- ▶ Alle Fenster gleich aufteilen: **E**.
- ▶ Kommentare toggeln: **C**.
- ▶ (Relative) Zeilennummern toggeln: **N**, **R**.
- ▶ Zeile farbig markieren: **c - 1**, ..., **c - 9**.
- ▶ Farbig markierte Zeilen verstecken / wieder einblenden: **H**.

Bedienung des PicoC-Compilers

Show-Mode, Teil 3

- ▶ Farbig markierte Zeilen entfernen **D**.
- ▶ Weiteres Fenster öffnen: **S**.

Makefile Bedienung

Show-Mode

- ▶ Starten für bestimmtes Programm:

```
> make show FILEPATH=<path-to-file> <more-options> .
```

- ▶ Starten für bestimmten Test in /tests:

```
> make test-show TESTNAME=<testname> <more-options> .
```

Makefile Bedienung

Makefile Optionen <more-options>

Kommandozeilenoption	Beschreibung	Standardwert
FILEPATH	Pfad zur Datei, die im Show-Mode angezeigt werden soll.	∅
TESTNAME	Name des Tests. Alles andere als der Basisname , wie die Dateiendung wird abgeschnitten.	∅
EXTENSION	Dateiendung , die an TESTNAME angehängt werden soll, damit daraus z.B. ./tests/TESTNAME.EXTENSION wird.	reti_states
NUM_WINDOWS	Anzahl Fenster auf die ein Dateiinhalt verteilt werden soll.	5
VERBOSE	Möglichkeit für eine ausführlichere Ausgabe die Kommandozeilenoption -v oder -vv zu aktivieren.	∅ bzw. -v für test-show
DEBUG	Möglichkeit die Kommandozeilenoption -d zu aktivieren, um bei make test-show TESTNAME=<testname> den Debugger für den entsprechenden Test <testname> zu starten.	∅

Tests

Bedienung

- ▶ Tests in /tests verifizieren und ausführen: `> make test <more-options> .`
 - ▶ `/run_tests.sh`, welches **zuerst** `/extract_input_and_expected.sh`, `/convert_to_c.py` und `/verify_tests.sh` ausführt.
- ▶ Tests vom GCC verifizieren lassen: `> make verify TESTNAME=<testname> .`
 - ▶ **vorher** `/extract_input_and_expected.sh`, `/convert_to_c.py` ausgeführt.
 - ▶ `/verify_tests.sh`.

Tests

Bedienung, Teil 2

► Informationen aus Tests extrahieren:

```
> make extract TESTNAME=<testname> .
```

► **Eingabe** // in:<space-sep-values> in <program>.in, **Ausgabe** // expected:<space-sep-values> in <program>.out_expected, **Datensegmentgröße** // datasegment:<size> **optional** in <program>.datasegment_size.

► `/extract_input_and_expected.sh` .

Tests

Bedienung, Teil 3

- ▶ Testdatei erstellen, die vom GCC kompiliert werden kann:

```
> make convert TESTNAME=<testname> .
```

- ▶ `input()`s werden durch **Eingaben** in `<program>.in` ersetzt.
- ▶ `print(exp)`s werden durch `#include<stdio.h>` und `printf("%d", exp)` ersetzt.
- ▶ `/convert_to_c.py` .

Tests

Testkategorien

Testkategorie	Beschreibung	Anzahl
basic	Grundlegende Funktionalitäten des PicoC-Compilers.	23
advanced	Spezialfälle und Kombinationen verschiedener Funktionalitäten des PicoC-Compilers.	20
hard	Lange und komplexe Tests, für welche die Funktionalitäten des PicoC-Compilers in perfekter Harmonie miteinander funktionieren müssen.	8
example	Bekannte Algorithmen, die als gutes, repräsentatives Beispiel für die Funktionsfähigkeit des PicoC-Compilers dienen.	24
error	Fehlermeldungen testen. Keine Verifikation wird ausgeführt.	69
exclude	Aufgrund vielfältiger Gründe soll keine Verifikation ausgeführt werden.	7
thesis	Codebeispiele der schriftlichen Ausarbeitung der Bachelorarbeit, die etwas umgeschrieben wurden, damit nicht nur das Durchlaufen dieser Tests getestet wird.	28
tobias	Vom Betreuer dieser Bachelorarbeit, M.Sc. Tobias Seufert geschrieben.	1

Codebeispiel

PicoC-Code

```
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 }
```

Lexikalische Analyse

Tokens generieren

PicoC-Code

```
// ...
struct stt (*pntr_on_ar_of_sts)[3][2]
↪ = &ar_of_sts;
// ...
```

Lexer

Tokenfolge

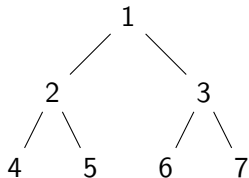
```
... Token('STRUCT', 'struct'),
↪ Token('NAME', 'stt'), Token('LPAR',
↪ '(', Token('MUL_DEREF_PNTR', '*'),
↪ Token('NAME', 'pntr_on_ar_of_sts'),
↪ Token('RPAR', ')'), Token('LSQB',
↪ '['), Token('NUM', '3'),
↪ Token('RSQB', ']'), Token('LSQB',
↪ '['), Token('NUM', '2'),
↪ Token('RSQB', ']'), Token('EQUAL',
↪ '='), Token('REF_AND', '&'),
↪ Token('NAME', 'ar_of_sts'),
↪ Token('SEMICOLON', ';'), ...
```

- ▶ ein „Zeiger auf ein Feld der Mächtigkeit 3 von Feldern der Mächtigkeit 2 von Verbunden des Typs stt“.

- ▶ *Clockwise/Spiral Rule*

Syntaktische Analyse

Darstellung von Bäumen

 \Rightarrow

Baum in der Darstellung
des PicoC-Compilers

```
1
 2
  4
  5
 3
  6
  7
```

- ▶ wächst von links-nach-rechts und alle Kinderknoten sind unter dem Elternknoten.

Syntaktische Analyse

Ableitungsbaum generieren

PicoC-Code

```
// ...
struct stt (*pntr_on_ar_of_sts)[3][2] =
↪ &ar_of_sts;
// ...
```

- ▶ ein „Zeiger auf ein Feld der Mächtigkeit 3 von Feldern der Mächtigkeit 2 von Verbunden des Typs stt“.

- ▶ *Clockwise/Spiral Rule*

Parser
& Lexer

Ableitungsbaum

```
...
init_stmt
  alloc
    type_spec
      struct_spec
        name      stt
      pntr_decl
        pntr_deg
          array_decl
            pntr_decl
              pntr_deg      *
              array_decl
                name pntr_on_ar_of_sts
                array_dims
                  array_dims
                    3
                    2
              initializer
            ...
```

Syntaktische Analyse

Ableitungsbaum vereinfachen

Ableitungsbaum

```

...
init_stmt
  alloc
    type_spec
      struct_spec
        name      stt
      pntr_decl
        pntr_deg
        array_decl
          pntr_decl
            pntr_deg      *
            array_decl
              name pntr_on_ar_of_sts
              array_dims
                3
                2
            initializer
            ...

```

Visitor

Vereinfachter Ableitungsbaum

```

...
init_stmt
  alloc
    pntr_decl
      pntr_deg
      array_decl
        array_dims
          3
          2
        pntr_decl
          pntr_deg *
          array_decl
            array_dims
            type_spec
              struct_spec
                name      stt
              name pntr_on_ar_of_sts
            initializer
            ...

```

Syntaktische Analyse

Abstrakten Syntaxbaum generieren

Vereinfachter Ableitungsbaum

```

...
init_stmt
  alloc
    pntr_decl
      pntr_deg
        array_decl
          array_dims
            3
            2
          pntr_decl
            pntr_deg *
            array_decl
              array_dims
              type_spec
              struct_spec
                name      stt
            name      pntr_on_ar_of_sts
          initializer
          ...

```

Transformer
ohne Umdrehen

Abstrakter Syntaxbaum

```

...
Assign
  Alloc
    Writeable,
    ArrayDecl
      [
        Num '3',
        Num '2'
      ],
    PtrDecl
      Num '1',
      StructSpec
        Name 'stt',
      Name 'pntr_on_ar_of_sts',
    Ref
      Name 'ar_of_sts',
    ...

```


Syntaktische Analyse

Abstrakten Syntaxbaum generieren

Vereinfachter Ableitungsbaum

```

...
init_stmt
  alloc
    pntr_decl
      pntr_deg
        array_decl
          array_dims
            3
            2
          pntr_decl
            pntr_deg *
            array_decl
              array_dims
              type_spec
              struct_spec
                name      stt
            name      pntr_on_ar_of_sts
          name      pntr_on_ar_of_sts
        initializer
        ...

```

Transformer
mit Umdrehen

Abstrakter Syntaxbaum

```

...
Assign
  Alloc
    Writeable,
    PntrDecl
      Num '1',
      ArrayDecl
        [
          Num '3',
          Num '2'
        ],
      StructSpec
        Name 'stt',
        Name 'pntr_on_ar_of_sts',
      Ref
        Name 'ar_of_sts',
    ...

```

Syntaktische Analyse

Abstrakten Syntaxbaum generieren

Tokenfolge

```
... Token('STRUCT', 'struct'), Token('NAME',
↳ 'stt'), Token('LPAR', '('),
↳ Token('MUL_DEREF_PNTR', '*'),
↳ Token('NAME', 'pntr_on_ar_of_sts'),
↳ Token('RPAR', ')'), Token('LSQB', '['),
↳ Token('NUM', '3'), Token('RSQB', ']'),
↳ Token('LSQB', '['), Token('NUM', '2'),
↳ Token('RSQB', ']'), Token('EQUAL', '='),
↳ Token('REF_AND', '&'), Token('NAME',
↳ 'ar_of_sts'), Token('SEMICOLON', ';'),
↳ ...
```

Syntaktische Analyse

Abstrakter Syntaxbaum

```
...
Assign
  Alloc
    Writeable,
    PntrDecl
      Num '1',
      ArrayDecl
        [
          Num '3',
          Num '2'
        ],
      StructSpec
        Name 'stt',
        Name 'pntr_on_ar_of_sts',
      Ref
        Name 'ar_of_sts',
    ...
```

Code Generierung

Abstrakter Syntaxbaum nach Syntaktischer Analyse

```
1 File
2   Name './example_presentation.ast',
3   [
4     StructDecl
5       Name 'stt',
6       [
7         Alloc(Writeable(), IntType('int'), Name('attr1'))
8         Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('attr2'))
9       ],
10    Exp
11      Alloc
12        Writeable,
13        ArrayDecl
14          [
15            Num '3',
16            Num '2'
17          ],
18        StructSpec
19          Name 'stt',
20          Name 'ar_of_sts',
```

Code Generierung

Abstrakter Syntaxbaum nach Syntaktischer Analyse, Teil 2

```
21 FunDef
22   IntType 'int',
23   Name 'fun',
24   [
25     Alloc(Writable(), PtrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt')))),
26     ↪ Name('param'))
27   ],
28   [
29     Assign(Subscr(Attr(Subscr(Deref(Deref(Name('param')), Num('0')), Num('2')), Num('1')), Name('attr2')),
30     ↪ Call(Name('input'), [])), Num('42'))
31     Return(Num('1'))
32   ],
33 ]
```

Code Generierung

Abstrakter Syntaxbaum nach Syntaktischer Analyse, Teil 3

```

31 FunDef
32   VoidType 'void',
33   Name 'main',
34   [],
35   [
36     Assign(Alloc(Writable(), PtrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt')))),
37     ↪ Name('pntr_on_ar_of_sts')), Ref(Name('ar_of_sts')))
38     Assign(Name('res'), Call(Name('fun'), [Name('pntr_on_ar_of_sts')])),
39     If
40     Name 'res',
41     [
42       Exp(Call(Name('print'), [Subscr(Attr(Subscr(Deref(Deref(Name('pntr_on_ar_of_sts')), Num('0')),
43       ↪ Num('2')), Num('1')), Name('attr2')), Num('1'))]))
44     ]
45   ]
46 ]

```

Code Generierung

PicoC-Shrink Pass

```
1 File
2   Name './example_presentation.picoc_shrink',
3   [
4     StructDecl
5       Name 'stt',
6       [
7         Alloc(Writeable(), IntType('int'), Name('attr1'))
8         Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('attr2'))
9       ],
10    Exp
11    Alloc
12    Writeable,
13    ArrayDecl
14    [
15      Num '3',
16      Num '2'
17    ],
18    StructSpec
19      Name 'stt',
20      Name 'ar_of_sts',
```

Code Generierung

PicoC-Shrink Pass, Teil 2

```
21 FunDef
22   IntType 'int',
23   Name 'fun',
24   [
25     Alloc(Writable(), PtrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt')))),
26     ↪ Name('param'))
27   ],
28   [
29     Assign(Subscr(Attr(Subscr(Subscr(Subscr(Name('param'), Num('0')), Num('2')), Num('1')), Name('attr2')),
30     ↪ Call(Name('input'), [])), Num('42'))
31     Return(Num('1'))
32   ],
```

Code Generierung

PicoC-Shrink Pass, Teil 3

```

31  FunDef
32      VoidType 'void',
33      Name 'main',
34      [],
35      [
36          Assign(Alloc(Writable(), PtrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt'))))),
37              ↪ Name('pntr_on_ar_of_sts')), Ref(Name('ar_of_sts')))
38          Assign(Name('res'), Call(Name('fun'), [Name('pntr_on_ar_of_sts')])),
39          If
40              Name 'res',
41              [
42                  Exp(Call(Name('print'), [Subscr(Attr(Subscr(Subscr(Name('pntr_on_ar_of_sts'), Num('0')),
43                      ↪ Num('2')), Num('1')), Name('attr2')), Num('1')))]))
44              ]
45      ]

```


Code Generierung

PicoC-Blocks Pass

```
1 File
2   Name './example_presentation.picoc_blocks',
3   [
4     StructDecl
5       Name 'stt',
6       [
7         Alloc(Writeable(), IntType('int'), Name('attr1'))
8         Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('attr2'))
9       ],
10    Exp
11    Alloc
12    Writeable,
13    ArrayDecl
14    [
15      Num '3',
16      Num '2'
17    ],
18    StructSpec
19      Name 'stt',
20      Name 'ar_of_sts',
```

Code Generierung

PicoC-Blocks Pass, Teil 2

```
21  FunDef
22      IntType 'int',
23      Name 'fun',
24      [
25          Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt')))),
26          ↪ Name('param'))
27      ],
28      [
29          Block
30              Name 'fun.3',
31              [
32                  Assign(Subscr(Attr(Subscr(Subscr(Subscr(Name('param'), Num('0')), Num('2')), Num('1')),
33                  ↪ Name('attr2')), Call(Name('input'), [])), Num('42'))
34                  Return(Num('1'))
35              ]
36      ],
```

Code Generierung

PicoC-Blocks Pass, Teil 3

```

35  FunDef
36      VoidType 'void',
37      Name 'main',
38      [],
39      [
40          Block
41              Name 'main.2',
42              [
43                  Assign(Alloc(Writable(), PtrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')],
44                      ↪ StructSpec(Name('stt')))), Name('ptr_on_ar_of_sts')), Ref(Name('ar_of_sts')))
45                  Assign(Alloc(Writable(), IntType('int'), Name('res')), Call(Name('fun'),
46                      ↪ [Name('ptr_on_ar_of_sts')]))
47                  // If(Name('res'), [],),
48                  IfElse
49                      Name 'res',
50                      [
51                          GoTo(Name('if.1'))
52                      ],
53                      [
54                          GoTo(Name('if_else_after.0'))
55                      ]
56              ],
57      ],

```

Code Generierung

PicoC-Blocks Pass, Teil 4

```
55     Block
56     Name 'if.1',
57     [
58         Exp(Call(Name('print'), [Subscr(Attr(Subscr(Subscr(Subscr(Name('pntr_on_ar_of_sts'), Num('0')),
59             ↪ Num('2')), Num('1')), Name('attr2')), Num('1'))]))
60         GoTo(Name('if_else_after.0'))
61     ],
62     Block
63     Name 'if_else_after.0',
64     []
65 ]
```

Code Generierung

PicoC-ANF Pass - Symboltabelle

```
1 SymbolTable
2   [
3     Symbol
4     {
5       type qualifier:      Empty()
6       datatype:            IntType('int')
7       name:                Name('attr1@stt')
8       value or address:    Empty()
9       position:            Pos(Num('5'), Num('16'))
10      size:                 Num('1')
11    },
12    Symbol
13    {
14      type qualifier:      Empty()
15      datatype:            ArrayDecl([Num('2')], IntType('int'))
16      name:                Name('attr2@stt')
17      value or address:    Empty()
18      position:            Pos(Num('5'), Num('27'))
19      size:                 Num('2')
20    },

```

Code Generierung

PicoC-ANF Pass - Symboltabelle, Teil 2

```

21 Symbol
22 {
23     type qualifier:      Empty()
24     datatype:            StructDecl(Name('stt'), [Alloc(Writable(), IntType('int'),
25 ↪ Name('attr1'))Alloc(Writable(), ArrayDecl([Num('2')], IntType('int')), Name('attr2'))])
26     name:                Name('stt')
27     value or address:    [Name('attr1@stt'), Name('attr2@stt')]
28     position:            Pos(Num('5'), Num('7'))
29     size:                Num('3')
30 },
31 Symbol
32 {
33     type qualifier:      Writable()
34     datatype:            ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt')))
35     name:                Name('ar_of_sts@global!')
36     value or address:    Num('0')
37     position:            Pos(Num('7'), Num('11'))
38     size:                Num('18')
39 },

```

Code Generierung

PicoC-ANF Pass - Symboltabelle, Teil 3

```

39 Symbol
40 {
41   type qualifier:      Empty()
42   datatype:            FunDecl(IntType('int'), Name('fun'), [Alloc(Writable()), PtrDecl(Num('1'),
43   ↪   ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt')))), Name('param'))])
44   name:                Name('fun')
45   value or address:    Empty()
46   position:            Pos(Num('9'), Num('4'))
47   size:                Empty()
48 },
49 Symbol
50 {
51   type qualifier:      Writable()
52   datatype:            PtrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt'))))
53   name:                Name('param@fun')
54   value or address:    Num('0')
55   position:            Pos(Num('9'), Num('21'))
56   size:                Num('1')
57 },

```

Code Generierung

PicoC-ANF Pass - Symboltabelle, Teil 4

```
57 Symbol
58 {
59     type qualifier:      Empty()
60     datatype:            FunDecl(VoidType('void'), Name('main'), [])
61     name:                 Name('main')
62     value or address:     Empty()
63     position:             Pos(Num('14'), Num('5'))
64     size:                 Empty()
65 },
66 Symbol
67 {
68     type qualifier:       Writeable()
69     datatype:              PtrDecl(Num('1'), ArrayDecl([Num('3'), Num('2')], StructSpec(Name('stt'))))
70     name:                  Name('ptr_on_ar_of_sts@main')
71     value or address:      Num('18')
72     position:              Pos(Num('15'), Num('15'))
73     size:                  Num('1')
74 },
```


Code Generierung

PicoC-ANF Pass - Symboltabelle, Teil 5

```
75 Symbol
76 {
77     type qualifier:    Writeable()
78     datatype:          IntType('int')
79     name:               Name('res@main')
80     value or address:   Num('19')
81     position:           Pos(Num('16'), Num('6'))
82     size:               Num('1')
83 }
84 ]
```

Code Generierung

PicoC-ANF Pass

```

1 File
2   Name './example_presentation.picoc_mon',
3   [
4     Block
5       Name 'global.4',
6       [],
7     Block
8       Name 'fun.3',
9       [
10        // Assign(Subscr(Attr(Subscr(Subscr(Subscr(Name('param'), Num('0')), Num('2')), Num('1')),
11        ↪ Name('attr2')), Call(Name('input'), [])), Num('42'))
12        Exp(Num('42'))
13        Ref(Stackframe(Num('0')))
14        Exp(Num('0'))
15        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
16        Exp(Num('2'))
17        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
18        Exp(Num('1'))
19        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
20        Ref(Attr(Stack(Num('1')), Name('attr2')))
21        Exp(Call(Name('input'), []))

```

Code Generierung

PicoC-ANF Pass, Teil 2

```
21     Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
22     Assign(Stack(Num('1')), Stack(Num('2')))
23     // Return(Num('1'))
24     Exp(Num('1'))
25     Return(Stack(Num('1')))
26 ],
27 Block
28   Name 'main.2',
29   [
30     // Assign(Name('pntr_on_ar_of_sts'), Ref(Name('ar_of_sts')))
31     Ref(Global(Num('0')))
32     Assign(Global(Num('18')), Stack(Num('1')))
33     // Assign(Name('res'), Call(Name('fun'), [Name('pntr_on_ar_of_sts')]))
34     StackMalloc(Num('2'))
35     Exp(Global(Num('18')))
36     NewStackframe(Name('fun.3'), GoTo(Name('addr@next_instr')))
37     Exp(GoTo(Name('fun.3')))
38     RemoveStackframe()
39     Exp(ACC)
40     Assign(Global(Num('19')), Stack(Num('1')))
```

Code Generierung

PicoC-ANF Pass, Teil 3

```
41 // If(Name('res'), [])
42 // IfElse(Name('res'), [], [])
43 Exp(Global(Num('19'))),
44 IfElse
45   Stack
46     Num '1',
47     [
48       GoTo(Name('if.1'))
49     ],
50     [
51       GoTo(Name('if_else_after.0'))
52     ]
53 ],
54 Block
55   Name 'if.1',
56   [
57     Ref(Global(Num('18')))
58     Exp(Num('0'))
59     Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
```

Code Generierung

PicoC-ANF Pass, Teil 4

```
60     Exp(Num('2'))
61     Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
62     Exp(Num('1'))
63     Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
64     Ref(Attr(Stack(Num('1')), Name('attr2')))
65     Exp(Num('1'))
66     Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
67     Exp(Stack(Num('1')))
68     Exp(Call(Name('print'), [Stack(Num('1'))]))
69     Exp(GoTo(Name('if_else_after.0')))
70 ],
71 Block
72   Name 'if_else_after.0',
73   [
74     Return(Empty())
75   ]
76 ]
```

Code Generierung

RETI-Blocks Pass

```
1 File
2   Name './example_presentation.reti_blocks',
3   [
4     Block
5       Name 'global.4',
6       [],
7     Block
8       Name 'fun.3',
9       [
10        # // Assign(Subscr(Attr(Subscr(Subscr(Subscr(Name('param'), Num('0')), Num('2')), Num('1')),
11        ↪ Name('attr2')), Call(Name('input'), [])), Num('42'))
12        # Exp(Num('42'))
13        SUBI SP 1;
14        LOADI ACC 42;
15        STOREIN SP ACC 1;
16        # Ref(Stackframe(Num('0')))
17        SUBI SP 1;
18        MOVE BAF IN1;
19        SUBI IN1 2;
20        STOREIN SP IN1 1;
21        # Exp(Num('0'))
22        SUBI SP 1;
```

Code Generierung

RETI-Blocks Pass, Teil 2

```
22     LOADI ACC 0;
23     STOREIN SP ACC 1;
24     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
25     LOADIN SP IN2 2;
26     LOADIN IN2 IN1 0;
27     LOADIN SP IN2 1;
28     MULTI IN2 18;
29     ADD IN1 IN2;
30     ADDI SP 1;
31     STOREIN SP IN1 1;
32     # Exp(Num('2'))
33     SUBI SP 1;
34     LOADI ACC 2;
35     STOREIN SP ACC 1;
36     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
37     LOADIN SP IN1 2;
38     LOADIN SP IN2 1;
39     MULTI IN2 6;
40     ADD IN1 IN2;
41     ADDI SP 1;
42     STOREIN SP IN1 1;
```

Code Generierung

RETI-Blocks Pass, Teil 3

```
43      # Exp(Num('1'))
44      SUBI SP 1;
45      LOADI ACC 1;
46      STOREIN SP ACC 1;
47      # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
48      LOADIN SP IN1 2;
49      LOADIN SP IN2 1;
50      MULTI IN2 3;
51      ADD IN1 IN2;
52      ADDI SP 1;
53      STOREIN SP IN1 1;
54      # Ref(Attr(Stack(Num('1')), Name('attr2')))
55      LOADIN SP IN1 1;
56      ADDI IN1 1;
57      STOREIN SP IN1 1;
58      CALL INPUT ACC;
59      SUBI SP 1;
60      STOREIN SP ACC 1;
61      # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
62      LOADIN SP IN1 2;
63      LOADIN SP IN2 1;
```


Code Generierung

RETI-Blocks Pass, Teil 4

```
64     MULTI IN2 1;
65     ADD IN1 IN2;
66     ADDI SP 1;
67     STOREIN SP IN1 1;
68     # Assign(Stack(Num('1')), Stack(Num('2')))
69     LOADIN SP IN1 1;
70     LOADIN SP ACC 2;
71     ADDI SP 2;
72     STOREIN IN1 ACC 0;
73     # // Return(Num('1'))
74     # Exp(Num('1'))
75     SUBI SP 1;
76     LOADI ACC 1;
77     STOREIN SP ACC 1;
78     # Return(Stack(Num('1')))
79     LOADIN SP ACC 1;
80     ADDI SP 1;
81     LOADIN BAF PC -1;
82 ],
83 Block
84     Name 'main.2',
```

Code Generierung

RETI-Blocks Pass, Teil 5

```
85  [
86    # // Assign(Name('pntr_on_ar_of_sts'), Ref(Name('ar_of_sts')))
87    # Ref(Global(Num('0')))
88    SUBI SP 1;
89    LOADI IN1 0;
90    ADD IN1 DS;
91    STOREIN SP IN1 1;
92    # Assign(Global(Num('18')), Stack(Num('1')))
93    LOADIN SP ACC 1;
94    STOREIN DS ACC 18;
95    ADDI SP 1;
96    # // Assign(Name('res'), Call(Name('fun'), [Name('pntr_on_ar_of_sts')]))
97    # StackMalloc(Num('2'))
98    SUBI SP 2;
99    # Exp(Global(Num('18')))
100   SUBI SP 1;
101   LOADIN DS ACC 18;
102   STOREIN SP ACC 1;
103   # NewStackframe(Name('fun.3'), GoTo(Name('addr@next_instr')))
104   MOVE BAF ACC;
105   ADDI SP 3;
```

Code Generierung

RETI-Blocks Pass, Teil 6

```
106     MOVE SP BAF;  
107     SUBI SP 3;  
108     STOREIN BAF ACC 0;  
109     LOADI ACC GoTo(Name('addr@next_instr'));  
110     ADD ACC CS;  
111     STOREIN BAF ACC -1;  
112     # Exp(GoTo(Name('fun.3')))  
113     Exp(GoTo(Name('fun.3')))  
114     # RemoveStackframe()  
115     MOVE BAF IN1;  
116     LOADIN IN1 BAF 0;  
117     MOVE IN1 SP;  
118     # Exp(ACC)  
119     SUBI SP 1;  
120     STOREIN SP ACC 1;  
121     # Assign(Global(Num('19')), Stack(Num('1')))  
122     LOADIN SP ACC 1;  
123     STOREIN DS ACC 19;  
124     ADDI SP 1;  
125     # // If(Name('res'), [])  
126     # // IfElse(Name('res'), [], [])
```

Code Generierung

RETI-Blocks Pass, Teil 7

```
127      # Exp(Global(Num('19')))  
128      SUBI SP 1;  
129      LOADIN DS ACC 19;  
130      STOREIN SP ACC 1;  
131      # IfElse(Stack(Num('1')), [], [])  
132      LOADIN SP ACC 1;  
133      ADDI SP 1;  
134      JUMP== GoTo(Name('if_else_after.0'));  
135      Exp(GoTo(Name('if.1')))  
136  ],  
137  Block  
138      Name 'if.1',  
139      [  
140          # Ref(Global(Num('18')))  
141          SUBI SP 1;  
142          LOADI IN1 18;  
143          ADD IN1 DS;  
144          STOREIN SP IN1 1;  
145          # Exp(Num('0'))  
146          SUBI SP 1;  
147          LOADI ACC 0;
```

Code Generierung

RETI-Blocks Pass, Teil 8

```
148     STOREIN SP ACC 1;
149     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
150     LOADIN SP IN2 2;
151     LOADIN IN2 IN1 0;
152     LOADIN SP IN2 1;
153     MULTI IN2 18;
154     ADD IN1 IN2;
155     ADDI SP 1;
156     STOREIN SP IN1 1;
157     # Exp(Num('2'))
158     SUBI SP 1;
159     LOADI ACC 2;
160     STOREIN SP ACC 1;
161     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
162     LOADIN SP IN1 2;
163     LOADIN SP IN2 1;
164     MULTI IN2 6;
165     ADD IN1 IN2;
166     ADDI SP 1;
167     STOREIN SP IN1 1;
168     # Exp(Num('1'))
```

Code Generierung

RETI-Blocks Pass, Teil 9

```
169     SUBI SP 1;
170     LOADI ACC 1;
171     STOREIN SP ACC 1;
172     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
173     LOADIN SP IN1 2;
174     LOADIN SP IN2 1;
175     MULTI IN2 3;
176     ADD IN1 IN2;
177     ADDI SP 1;
178     STOREIN SP IN1 1;
179     # Ref(Attr(Stack(Num('1')), Name('attr2')))
180     LOADIN SP IN1 1;
181     ADDI IN1 1;
182     STOREIN SP IN1 1;
183     # Exp(Num('1'))
184     SUBI SP 1;
185     LOADI ACC 1;
186     STOREIN SP ACC 1;
187     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
188     LOADIN SP IN1 2;
189     LOADIN SP IN2 1;
```

Code Generierung

RETI-Blocks Pass, Teil 10

```
190     MULTI IN2 1;
191     ADD IN1 IN2;
192     ADDI SP 1;
193     STOREIN SP IN1 1;
194     # Exp(Stack(Num('1')))
195     LOADIN SP IN1 1;
196     LOADIN IN1 ACC 0;
197     STOREIN SP ACC 1;
198     LOADIN SP ACC 1;
199     ADDI SP 1;
200     CALL PRINT ACC;
201     # Exp(GoTo(Name('if_else_after.0')))
202     Exp(GoTo(Name('if_else_after.0')))
203 ],
204 Block
205   Name 'if_else_after.0',
206   [
207     # Return(Empty())
208     LOADIN BAF PC -1;
209   ]
210 ]
```

Code Generierung

RETI-Patch Pass

```
1 File
2   Name './example_presentation.reti_patch',
3   [
4     Block
5       Name 'global.4',
6       [
7         # // Exp(GoTo(Name('main.2')))
8         Exp(GoTo(Name('main.2')))
9       ],
10    Block
11      Name 'fun.3',
12      [
13        # // Assign(Subscr(Attr(Subscr(Subscr(Subscr(Name('param'), Num('0')), Num('2')), Num('1')),
14        ↪ Name('attr2')), Call(Name('input'), [])), Num('42'))
15        # Exp(Num('42'))
16        SUBI SP 1;
17        LOADI ACC 42;
18        STOREIN SP ACC 1;
19        # Ref(Stackframe(Num('0')))
20        SUBI SP 1;
21        MOVE BAF IN1;
22        SUBI IN1 2;
23        STOREIN SP IN1 1;
```


Code Generierung

RETI-Patch Pass, Teil 2

```
23      # Exp(Num('0'))
24      SUBI SP 1;
25      LOADI ACC 0;
26      STOREIN SP ACC 1;
27      # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
28      LOADIN SP IN2 2;
29      LOADIN IN2 IN1 0;
30      LOADIN SP IN2 1;
31      MULTI IN2 18;
32      ADD IN1 IN2;
33      ADDI SP 1;
34      STOREIN SP IN1 1;
35      # Exp(Num('2'))
36      SUBI SP 1;
37      LOADI ACC 2;
38      STOREIN SP ACC 1;
39      # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
40      LOADIN SP IN1 2;
41      LOADIN SP IN2 1;
42      MULTI IN2 6;
43      ADD IN1 IN2;
44      ADDI SP 1;
```

Code Generierung

RETI-Patch Pass, Teil 3

```
45     STOREIN SP IN1 1;
46     # Exp(Num('1'))
47     SUBI SP 1;
48     LOADI ACC 1;
49     STOREIN SP ACC 1;
50     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
51     LOADIN SP IN1 2;
52     LOADIN SP IN2 1;
53     MULTI IN2 3;
54     ADD IN1 IN2;
55     ADDI SP 1;
56     STOREIN SP IN1 1;
57     # Ref(Attr(Stack(Num('1')), Name('attr2')))
58     LOADIN SP IN1 1;
59     ADDI IN1 1;
60     STOREIN SP IN1 1;
61     CALL INPUT ACC;
62     SUBI SP 1;
63     STOREIN SP ACC 1;
64     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
65     LOADIN SP IN1 2;
66     LOADIN SP IN2 1;
```

Code Generierung

RETI-Patch Pass, Teil 4

```
67     MULTI IN2 1;
68     ADD IN1 IN2;
69     ADDI SP 1;
70     STOREIN SP IN1 1;
71     # Assign(Stack(Num('1')), Stack(Num('2')))
72     LOADIN SP IN1 1;
73     LOADIN SP ACC 2;
74     ADDI SP 2;
75     STOREIN IN1 ACC 0;
76     # // Return(Num('1'))
77     # Exp(Num('1'))
78     SUBI SP 1;
79     LOADI ACC 1;
80     STOREIN SP ACC 1;
81     # Return(Stack(Num('1')))
82     LOADIN SP ACC 1;
83     ADDI SP 1;
84     LOADIN BAF PC -1;
85 ],
86 Block
87   Name 'main.2',
88   [
```

Code Generierung

RETI-Patch Pass, Teil 5

```
89      # // Assign(Name('pntr_on_ar_of_sts'), Ref(Name('ar_of_sts'))))
90      # Ref(Global(Num('0'))))
91      SUBI SP 1;
92      LOADI IN1 0;
93      ADD IN1 DS;
94      STOREIN SP IN1 1;
95      # Assign(Global(Num('18')), Stack(Num('1'))))
96      LOADIN SP ACC 1;
97      STOREIN DS ACC 18;
98      ADDI SP 1;
99      # // Assign(Name('res'), Call(Name('fun'), [Name('pntr_on_ar_of_sts')]))
100     # StackMalloc(Num('2'))
101     SUBI SP 2;
102     # Exp(Global(Num('18'))))
103     SUBI SP 1;
104     LOADIN DS ACC 18;
105     STOREIN SP ACC 1;
106     # NewStackframe(Name('fun.3'), GoTo(Name('addr@next_instr'))))
107     MOVE BAF ACC;
108     ADDI SP 3;
109     MOVE SP BAF;
110     SUBI SP 3;
```

Code Generierung

RETI-Patch Pass, Teil 6

```
111     STOREIN BAF ACC 0;
112     LOADI ACC GoTo(Name('addr@next_instr'));
113     ADD ACC CS;
114     STOREIN BAF ACC -1;
115     # Exp(GoTo(Name('fun.3')))
116     Exp(GoTo(Name('fun.3')))
117     # RemoveStackframe()
118     MOVE BAF IN1;
119     LOADIN IN1 BAF 0;
120     MOVE IN1 SP;
121     # Exp(ACC)
122     SUBI SP 1;
123     STOREIN SP ACC 1;
124     # Assign(Global(Num('19')), Stack(Num('1')))
125     LOADIN SP ACC 1;
126     STOREIN DS ACC 19;
127     ADDI SP 1;
128     # // If(Name('res'), [])
129     # // IfElse(Name('res'), [], [])
130     # Exp(Global(Num('19')))
131     SUBI SP 1;
132     LOADIN DS ACC 19;
```

Code Generierung

RETI-Patch Pass, Teil 7

```
133     STOREIN SP ACC 1;
134     # IfElse(Stack(Num('1')), [], [])
135     LOADIN SP ACC 1;
136     ADDI SP 1;
137     JUMP== GoTo(Name('if_else_after.0'));
138     # // not included Exp(GoTo(Name('if.1')))
139 ],
140 Block
141   Name 'if.1',
142   [
143     # Ref(Global(Num('18')))
144     SUBI SP 1;
145     LOADI IN1 18;
146     ADD IN1 DS;
147     STOREIN SP IN1 1;
148     # Exp(Num('0'))
149     SUBI SP 1;
150     LOADI ACC 0;
151     STOREIN SP ACC 1;
152     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
153     LOADIN SP IN2 2;
154     LOADIN IN2 IN1 0;
```

Code Generierung

RETI-Patch Pass, Teil 8

```
155     LOADIN SP IN2 1;
156     MULTI IN2 18;
157     ADD IN1 IN2;
158     ADDI SP 1;
159     STOREIN SP IN1 1;
160     # Exp(Num('2'))
161     SUBI SP 1;
162     LOADI ACC 2;
163     STOREIN SP ACC 1;
164     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
165     LOADIN SP IN1 2;
166     LOADIN SP IN2 1;
167     MULTI IN2 6;
168     ADD IN1 IN2;
169     ADDI SP 1;
170     STOREIN SP IN1 1;
171     # Exp(Num('1'))
172     SUBI SP 1;
173     LOADI ACC 1;
174     STOREIN SP ACC 1;
175     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
176     LOADIN SP IN1 2;
```

Code Generierung

RETI-Patch Pass, Teil 9

```
177     LOADIN SP IN2 1;
178     MULTI IN2 3;
179     ADD IN1 IN2;
180     ADDI SP 1;
181     STOREIN SP IN1 1;
182     # Ref(Attr(Stack(Num('1')), Name('attr2')))
183     LOADIN SP IN1 1;
184     ADDI IN1 1;
185     STOREIN SP IN1 1;
186     # Exp(Num('1'))
187     SUBI SP 1;
188     LOADI ACC 1;
189     STOREIN SP ACC 1;
190     # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
191     LOADIN SP IN1 2;
192     LOADIN SP IN2 1;
193     MULTI IN2 1;
194     ADD IN1 IN2;
195     ADDI SP 1;
196     STOREIN SP IN1 1;
197     # Exp(Stack(Num('1')))
198     LOADIN SP IN1 1;
```


Code Generierung

RETI-Patch Pass, Teil 10

```
199     LOADIN IN1 ACC 0;
200     STOREIN SP ACC 1;
201     LOADIN SP ACC 1;
202     ADDI SP 1;
203     CALL PRINT ACC;
204     # Exp(GoTo(Name('if_else_after.0')))
205     # // not included Exp(GoTo(Name('if_else_after.0')))
206 ],
207 Block
208   Name 'if_else_after.0',
209   [
210     # Return(Empty())
211     LOADIN BAF PC -1;
212   ]
213 ]
```

Code Generierung

RETI Pass

```
1 # // Exp(GoTo(Name('main.2')))  
2 JUMP 58;  
3 # // Assign(Subscr(Attr(Subscr(Subscr(Subscr(Name('param'), Num('0')), Num('2')), Num('1')), Name('attr2')),  
↪ Call(Name('input'), [])), Num('42'))  
4 # Exp(Num('42'))  
5 SUBI SP 1;  
6 LOADI ACC 42;  
7 STOREIN SP ACC 1;  
8 # Ref(Stackframe(Num('0')))  
9 SUBI SP 1;  
10 MOVE BAF IN1;  
11 SUBI IN1 2;  
12 STOREIN SP IN1 1;  
13 # Exp(Num('0'))  
14 SUBI SP 1;  
15 LOADI ACC 0;  
16 STOREIN SP ACC 1;  
17 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))  
18 LOADIN SP IN2 2;  
19 LOADIN IN2 IN1 0;  
20 LOADIN SP IN2 1;  
21 MULTI IN2 18;
```

Code Generierung

RETI Pass, Teil 2

```
22 ADD IN1 IN2;
23 ADDI SP 1;
24 STOREIN SP IN1 1;
25 # Exp(Num('2'))
26 SUBI SP 1;
27 LOADI ACC 2;
28 STOREIN SP ACC 1;
29 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
30 LOADIN SP IN1 2;
31 LOADIN SP IN2 1;
32 MULTI IN2 6;
33 ADD IN1 IN2;
34 ADDI SP 1;
35 STOREIN SP IN1 1;
36 # Exp(Num('1'))
37 SUBI SP 1;
38 LOADI ACC 1;
39 STOREIN SP ACC 1;
40 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
41 LOADIN SP IN1 2;
42 LOADIN SP IN2 1;
```

Code Generierung

RETI Pass, Teil 3

```
43 MULTI IN2 3;
44 ADD IN1 IN2;
45 ADDI SP 1;
46 STOREIN SP IN1 1;
47 # Ref(Attr(Stack(Num('1')), Name('attr2')))
48 LOADIN SP IN1 1;
49 ADDI IN1 1;
50 STOREIN SP IN1 1;
51 CALL INPUT ACC;
52 SUBI SP 1;
53 STOREIN SP ACC 1;
54 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
55 LOADIN SP IN1 2;
56 LOADIN SP IN2 1;
57 MULTI IN2 1;
58 ADD IN1 IN2;
59 ADDI SP 1;
60 STOREIN SP IN1 1;
61 # Assign(Stack(Num('1')), Stack(Num('2')))
62 LOADIN SP IN1 1;
63 LOADIN SP ACC 2;
```

Code Generierung

RETI Pass, Teil 4

```
64 ADDI SP 2;
65 STOREIN IN1 ACC 0;
66 # // Return(Num('1'))
67 # Exp(Num('1'))
68 SUBI SP 1;
69 LOADI ACC 1;
70 STOREIN SP ACC 1;
71 # Return(Stack(Num('1')))
72 LOADIN SP ACC 1;
73 ADDI SP 1;
74 LOADIN BAF PC -1;
75 # // Assign(Name('pntr_on_ar_of_sts'), Ref(Name('ar_of_sts')))
76 # Ref(Global(Num('0')))
77 SUBI SP 1;
78 LOADI IN1 0;
79 ADD IN1 DS;
80 STOREIN SP IN1 1;
81 # Assign(Global(Num('18')), Stack(Num('1')))
82 LOADIN SP ACC 1;
83 STOREIN DS ACC 18;
84 ADDI SP 1;
```

Code Generierung

RETI Pass, Teil 5

```
85 # // Assign(Name('res'), Call(Name('fun'), [Name('pntr_on_ar_of_sts')]))
86 # StackMalloc(Num('2'))
87 SUBI SP 2;
88 # Exp(Global(Num('18'))))
89 SUBI SP 1;
90 LOADIN DS ACC 18;
91 STOREIN SP ACC 1;
92 # NewStackframe(Name('fun.3'), GoTo(Name('addr@next_instr'))))
93 MOVE BAF ACC;
94 ADDI SP 3;
95 MOVE SP BAF;
96 SUBI SP 3;
97 STOREIN BAF ACC 0;
98 LOADI ACC 78;
99 ADD ACC CS;
100 STOREIN BAF ACC -1;
101 # Exp(GoTo(Name('fun.3'))))
102 JUMP -76;
103 # RemoveStackframe()
104 MOVE BAF IN1;
105 LOADIN IN1 BAF 0;
```

Code Generierung

RETI Pass, Teil 6

```
106 MOVE IN1 SP;
107 # Exp(ACC)
108 SUBI SP 1;
109 STOREIN SP ACC 1;
110 # Assign(Global(Num('19')), Stack(Num('1')))
111 LOADIN SP ACC 1;
112 STOREIN DS ACC 19;
113 ADDI SP 1;
114 # // If(Name('res'), [])
115 # // IfElse(Name('res'), [], [])
116 # Exp(Global(Num('19')))
117 SUBI SP 1;
118 LOADIN DS ACC 19;
119 STOREIN SP ACC 1;
120 # IfElse(Stack(Num('1')), [], [])
121 LOADIN SP ACC 1;
122 ADDI SP 1;
123 JUMP== 51;
124 # // not included Exp(GoTo(Name('if.1')))
125 # Ref(Global(Num('18')))
126 SUBI SP 1;
```

Code Generierung

RETI Pass, Teil 7

```
127 LOADI IN1 18;
128 ADD IN1 DS;
129 STOREIN SP IN1 1;
130 # Exp(Num('0'))
131 SUBI SP 1;
132 LOADI ACC 0;
133 STOREIN SP ACC 1;
134 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
135 LOADIN SP IN2 2;
136 LOADIN IN2 IN1 0;
137 LOADIN SP IN2 1;
138 MULTI IN2 18;
139 ADD IN1 IN2;
140 ADDI SP 1;
141 STOREIN SP IN1 1;
142 # Exp(Num('2'))
143 SUBI SP 1;
144 LOADI ACC 2;
145 STOREIN SP ACC 1;
146 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
147 LOADIN SP IN1 2;
```


Code Generierung

RETI Pass, Teil 8

```
148 LOADIN SP IN2 1;
149 MULTI IN2 6;
150 ADD IN1 IN2;
151 ADDI SP 1;
152 STOREIN SP IN1 1;
153 # Exp(Num('1'))
154 SUBI SP 1;
155 LOADI ACC 1;
156 STOREIN SP ACC 1;
157 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
158 LOADIN SP IN1 2;
159 LOADIN SP IN2 1;
160 MULTI IN2 3;
161 ADD IN1 IN2;
162 ADDI SP 1;
163 STOREIN SP IN1 1;
164 # Ref(Attr(Stack(Num('1')), Name('attr2')))
165 LOADIN SP IN1 1;
166 ADDI IN1 1;
167 STOREIN SP IN1 1;
168 # Exp(Num('1'))
```

Code Generierung

RETI Pass, Teil 9

```
169 SUBI SP 1;
170 LOADI ACC 1;
171 STOREIN SP ACC 1;
172 # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
173 LOADIN SP IN1 2;
174 LOADIN SP IN2 1;
175 MULTI IN2 1;
176 ADD IN1 IN2;
177 ADDI SP 1;
178 STOREIN SP IN1 1;
179 # Exp(Stack(Num('1')))
180 LOADIN SP IN1 1;
181 LOADIN IN1 ACC 0;
182 STOREIN SP ACC 1;
183 LOADIN SP ACC 1;
184 ADDI SP 1;
185 CALL PRINT ACC;
186 # Exp(GoTo(Name('if_else_after.0')))
187 # // not included Exp(GoTo(Name('if_else_after.0')))
188 # Return(Empty())
189 LOADIN BAF PC -1;
```

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



Clockwise/Spiral Rule. URL:

<https://c-faq.com/decl/spiral.anderson.html> (besucht am 29.07.2022).