Albert Ludwigs Universität Freiburg

Technische Fakultät

PicoC-Compiler

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

Bachelorarbeit

Abgabedatum: 28th April 2022

Author:
Jürgen Mattheis

Gutachter:
Prof. Dr. Scholl

Betreung:
M.Sc. Seufert

Eine Bachelorarbeit am Lehrstuhl für
Betriebssysteme

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

Grammatikverzeichnis

0.0.1 Umsetzung von Pointern

0.0.1.1 Referenzierung

Die **Referenzierung** `&<var>` wird im Folgenden anhand des Beispiels in Code 0.1 erklärt.

```
1 void main() {
2   int var = 42;
3   int *pntr = &var;
4 }
```

Code 0.1: PicoC Code für Pointer Referenzierung

Der Knoten `Ref(Name('var'))` repräsentiert im **Abstrakt Syntax Tree** in Code 0.2 eine **Referenzierung** `&<var>`.

```
1 File
2   Name './example_pntr_ref.ast',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
10        Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr')),
          ↪  Ref(Name('var')))
11      ]
12  ]
```

Code 0.2: Abstract Syntax Tree für Pointer Referenzierung

Im **PicoC-Mon Pass** in Code 0.3 wird der Knoten `Ref(Name('var'))` durch die Knoten `Ref(GlobalRead(Num('0')))` und `Assign(GlobalWrite(Num('1')), Tmp(Num('1')))` ersetzt. Im Fall, dass in `Ref(exp))` das `exp` vielleicht nicht direkt ein `Name('var')` enthält und `exp` z.B. ein `Subscr(Attr(Name('var')))` ist, sind noch weitere Anweisungen zwischen den Zeilen `11` und `12` nötig, die sich in diesem Beispiel um das Übersetzen von `Subscr(exp)` und `Attr(exp)` nach dem Schema in Subkapitel 0.0.4.2 kümmern.

```
1 File
2   Name './example_pntr_ref.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('var'), Num('42'))
8         Exp(Num('42'))
9         Assign(Global(Num('0')), Stack(Num('1')))
10        // Assign(Name('pntr'), Ref(Name('var')))
11        Ref(Global(Num('0')))
```

```
12        Assign(Global(Num('1')), Stack(Num('1')))
13        Return(Empty())
14      ]
15  ]
```

Code 0.3: PicoC Mon Pass für Pointer Referenzierung

Im **PicoC-Blocks Pass** in Code 0.4 werden die **PicoC-Knoten** `Ref(Global(Num('0')))` und `Assign(Global(Num('1')), Stack(Num('1')))` durch ihre entsprechenden **RETI-Knoten** ersetzt.

```
1 File
2   Name './example_pntr_ref.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('var'), Num('42'))
8         # Exp(Num('42'))
9         SUBI SP 1;
10        LOADI ACC 42;
11        STOREIN SP ACC 1;
12        # Assign(Global(Num('0')), Stack(Num('1')))
13        LOADIN SP ACC 1;
14        STOREIN DS ACC 0;
15        ADDI SP 1;
16        # // Assign(Name('pntr'), Ref(Name('var')))
17        # Ref(Global(Num('0')))
18        SUBI SP 1;
19        LOADI IN1 0;
20        ADD IN1 DS;
21        STOREIN SP IN1 1;
22        # Assign(Global(Num('1')), Stack(Num('1')))
23        LOADIN SP ACC 1;
24        STOREIN DS ACC 1;
25        ADDI SP 1;
26        # Return(Empty())
27        LOADIN BAF PC -1;
28      ]
29  ]
```

Code 0.4: RETI Blocks Pass für Pointer Referenzierung

0.0.1.2 Dereferenzierung durch Zugriff auf Arrayindex ersetzen

Die **Dereferenzierung** `*<var>` wird im Folgenden anhand des Beispiels in Code 0.5 erklärt.

```
1 void main() {
2   int var = 42;
3   int *pntr = &var;
4   *pntr;
5 }
```

Code 0.5: PicoC Code für Pointer Dereferenzierung

Der Knoten `Deref(Name('var'))` repräsentiert im **Abstrakt Syntax Tree** in Code 0.6 eine **Dereferenzierung** `*<var>`.

```
File
  Name './example_pntr_deref.ast',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
        Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr')),
        ↪  Ref(Name('var')))
        Exp(Deref(Name('pntr'), Num('0')))
      ]
  ]
```

Code 0.6: Abstract Syntax Tree für Pointer Dereferenzierung

Im **PicoC-Shrink Pass** in Code 0.7 wird ein Trick angewandet, bei dem jeder Knoten `Deref(Name('pntr'), Num('0'))` einfach durch den Knoten `Subscr(Name('pntr'), Num('0'))` ersetzt wird. Der Trick besteht darin, dass der **Dereferenzoperator** `*(<var> + <i>)` sich identisch zum **Operator für den Zugriff auf einen Arrayindex** `<var>[<i>]` verhält[1]. Damit sparrt man sich viele vermeidbare **Fallunterscheidungen** und **doppelten Code** und kann die **Derefenzierung** `*(<var> + <i>)` einfach von den Routinen für einen **Zugriff auf einen Arrayindex** `<var>[<i>]` übernehmen lassen.

```
File
  Name './example_pntr_deref.picoc_shrink',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
        Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr')),
        ↪  Ref(Name('var')))
        Exp(Subscr(Name('pntr'), Num('0')))
      ]
  ]
```

Code 0.7: PicoC Shrink Pass für Pointer Dereferenzierung

[1] In der Sprache L_C gibt es einen Unterschied bei der Initialisierung bei z.B. `<datatype> *<var> = "string"` und `<datatype> <var>[<i>] = "string"`, der allerdings nichts mit den beiden Operatoren zu tuen hat, sondern mit der **Initialisierung**, bei der die Sprache L_C verwirrenderweise die eckigen Klammern `[]` genauso, wie beim **Operator für den Zugriff auf einen Arrayindex**, vor den Bezeichner schreibt: `<var>[<i>]`, obwohl es ein **Derived Datatype** ist.

0.0.2 Umsetzung von Arrays

0.0.2.1 Initialisierung von Arrays

Die **Initialisierung** eines **Arrays** (`<datatype> <var>[2][1] = {{3+1}, {4}}`) wird im Folgenden anhand des Beispiels in Code 0.8 erklärt.

```
1 void main() {
2   int ar[2][1] = {{3+1}, {4}};
3 }
4
5 void fun() {
6   int ar[2][2] = {{3, 4}, {5, 6}};
7 }
```

Code 0.8: PicoC Code für Array Initialisierung

Die **Initialisierung** eines **Arrays** `<datatype> <var>[2][1] = {{3+1}, {4}}` wird im **Abstrakt Syntax Tree** in Code 0.9 mithilfe der Komposition `Assign(Alloc(Writeable(), ArrayDecl([Num('2'), Num('1')], IntType('int')), Name('ar')), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]), Array([Num('4')])]))` dargestellt.

```
1 File
2   Name './example_array_init.ast',
3   [
4     FunDef
5       VoidType 'void',
6       Name 'main',
7       [],
8       [
9         Assign(Alloc(Writeable(), ArrayDecl([Num('2'), Num('1')], IntType('int')),
          ↪  Name('ar')), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]),
          ↪  Array([Num('4')])]))
10      ],
11    FunDef
12      VoidType 'void',
13      Name 'fun',
14      [],
15      [
16        Assign(Alloc(Writeable(), ArrayDecl([Num('2'), Num('2')], IntType('int')),
          ↪  Name('ar')), Array([Array([Num('3'), Num('4')]), Array([Num('5'), Num('6')])]))
17      ]
18  ]
```

Code 0.9: Abstract Syntax Tree für Array Initialisierung

Bei der **Initialisierung** eines **Arrays** wird zuerst `Alloc(Writeable(), ArrayDecl([Num('2'), Num('1')], IntType('int')))` ausgewertet, da eine Variable zuerst definiert sein muss, bevor man sie verwenden kann[2]. Das **Definieren** der Variable `ar` erfolgt mittels der **Symboltabelle**, die in Code 0.10 dargestellt ist.

[2] Das Widerspricht der üblichen Auswertungsreihenfolge beim **Zuweisungsoperator** `=`, der **rechtsassoziativ** ist. Der **Zuweisungsoperator** `=` tritt allerdings erst später in Aktion.

Bei Variablen auf dem **Stackframe** wird ein Array **rückwärts** auf das Stackframe geschrieben und auch die **Adresse des ersten Elements** als Adresse des Arrays genommen. Dies macht den **Zugriff auf ein Arrayelement** in Subkapitel 0.0.2.2 deutlich unkomplizierter, da man so nicht mehr zwischen **Stackframe** und **Globalen Statischen Daten** beim **Zugriff auf ein Arrayelement** unterscheiden muss, da es Probleme macht, dass ein **Stackframe** in die Entgegengesetzt Richtung der **Globalen Statischen Daten** wächst[3].

```
SymbolTable
  [
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('main'), [])
        name:                   Name('main')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('5'))
        size:                   Empty()
      },
    Symbol
      {
        type qualifier:         Writeable()
        datatype:               ArrayDecl([Num('2'), Num('1')], IntType('int'))
        name:                   Name('ar@main')
        value or address:       Num('0')
        position:               Pos(Num('2'), Num('6'))
        size:                   Num('2')
      },
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('fun'), [])
        name:                   Name('fun')
        value or address:       Empty()
        position:               Pos(Num('5'), Num('5'))
        size:                   Empty()
      },
    Symbol
      {
        type qualifier:         Writeable()
        datatype:               ArrayDecl([Num('2'), Num('2')], IntType('int'))
        name:                   Name('ar@fun')
        value or address:       Num('3')
        position:               Pos(Num('6'), Num('6'))
        size:                   Num('4')
      }
  ]
```

Code 0.10: Symboltabelle für Array Initialisierung

Im **PiocC-Mon Pass** in Code 0.11 werden zuerst die **Ausdrücke** im **Array-Initializer** `Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]), Array([Num('4')])])` nach dem **Depth-First-Search** Schema, von **links-nach-rechts** ausgewertet und auf den **Stack** geschrieben.

[3] Wenn man beim **GCC** *GCC, the GNU Compiler Collection - GNU Project* einen Stackframe mittels des **GDB** *GCC, the GNU Compiler Collection - GNU Project* beobachtet, sieht man, dass dieser es genauso macht.

Im finalen Schritt muss zwischen **Globalen Statischen Daten** bei der `main`-Funktion und **Stackframe** bei der Funktion `fun` unterschieden werden. Die auf den Stack ausgewerteten Expressions werden mittels der Komposition `Assign(Global(Num('0')), Stack(Num('2')))` bzw. `Assign(Stackframe(Num('3')), Stack(Num('4')))`, versetzt in der selben Reihenfolge zu den **Globalen Statischen Daten** bzw. auf den **Stackframe** geschrieben.

In die Knoten `Global('0')` und `Stackframe('3')` wurde hierbei die **Startadresse** des jeweiligen Arrays geschrieben, sodass man nach dem **PicoC-Mon Pass** nie mehr Variablen in der **Symboltabelle** nachsehen muss und gleich weiß, ob sie bei den **Globalen Statischen Daten** oder auf dem **Stackframe** liegen.

```
File
  Name './example_array_init.picoc_mon',
  [
    Block
      Name 'main.1',
      [
        // Assign(Name('ar'), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]),
        ↪  Array([Num('4')])]))
        Exp(Num('3'))
        Exp(Num('1'))
        Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
        Exp(Num('4'))
        Assign(Global(Num('0')), Stack(Num('2')))
        Return(Empty())
      ],
    Block
      Name 'fun.0',
      [
        // Assign(Name('ar'), Array([Array([Num('3'), Num('4')]), Array([Num('5'),
        ↪  Num('6')])]))
        Exp(Num('3'))
        Exp(Num('4'))
        Exp(Num('5'))
        Exp(Num('6'))
        Assign(Stackframe(Num('3')), Stack(Num('4')))
        Return(Empty())
      ]
  ]
```

Code 0.11: PicoC Mon Pass für Array Initialisierung

Im **PicoC-Blocks Pass** in Code 0.12 werden die **PicoC-Knoten** für die Ausdrücke `Exp(exp)` und `Assign(Global(Num('0')), Stack(Num('2')))` bzw. `Assign(Stackframe(Num('3')), Stack(Num('4')))` durch ihre entsprechenden **RETI-Knoten** ersetzt.

```
File
  Name './example_array_init.reti_blocks',
  [
    Block
      Name 'main.1',
      [
        # // Assign(Name('ar'), Array([Array([BinOp(Num('3'), Add('+'), Num('1'))]),
        ↪  Array([Num('4')])]))
```

```
 8         # Exp(Num('3'))
 9         SUBI SP 1;
10         LOADI ACC 3;
11         STOREIN SP ACC 1;
12         # Exp(Num('1'))
13         SUBI SP 1;
14         LOADI ACC 1;
15         STOREIN SP ACC 1;
16         # Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
17         LOADIN SP ACC 2;
18         LOADIN SP IN2 1;
19         ADD ACC IN2;
20         STOREIN SP ACC 2;
21         ADDI SP 1;
22         # Exp(Num('4'))
23         SUBI SP 1;
24         LOADI ACC 4;
25         STOREIN SP ACC 1;
26         # Assign(Global(Num('0')), Stack(Num('2')))
27         LOADIN SP ACC 1;
28         STOREIN DS ACC 1;
29         LOADIN SP ACC 2;
30         STOREIN DS ACC 0;
31         ADDI SP 2;
32         # Return(Empty())
33         LOADIN BAF PC -1;
34       ],
35     Block
36       Name 'fun.0',
37       [
38         # // Assign(Name('ar'), Array([Array([Num('3'), Num('4')]), Array([Num('5'),
           ↪  Num('6')])]))
39         # Exp(Num('3'))
40         SUBI SP 1;
41         LOADI ACC 3;
42         STOREIN SP ACC 1;
43         # Exp(Num('4'))
44         SUBI SP 1;
45         LOADI ACC 4;
46         STOREIN SP ACC 1;
47         # Exp(Num('5'))
48         SUBI SP 1;
49         LOADI ACC 5;
50         STOREIN SP ACC 1;
51         # Exp(Num('6'))
52         SUBI SP 1;
53         LOADI ACC 6;
54         STOREIN SP ACC 1;
55         # Assign(Stackframe(Num('3')), Stack(Num('4')))
56         LOADIN SP ACC 1;
57         STOREIN BAF ACC -2;
58         LOADIN SP ACC 2;
59         STOREIN BAF ACC -3;
60         LOADIN SP ACC 3;
61         STOREIN BAF ACC -4;
62         LOADIN SP ACC 4;
63         STOREIN BAF ACC -5;
```

```
64          ADDI SP 4;
65          # Return(Empty())
66          LOADIN BAF PC -1;
67        ]
68    ]
```

Code 0.12: RETI Blocks Pass für Array Initialisierung

0.0.2.2 Zugriff auf ein Arrayelement

Der **Zugriff auf ein Arrayelement** `ar[0]` wird im Folgenden anhand des Beispiels in Code 0.13 erklärt.

```
1  void main() {
2    int ar[1] = {42};
3    ar[0];
4  }
5
6  void fun() {
7    int ar[3] = {1, 2, 3};
8    ar[1+1];
9  }
```

Code 0.13: PicoC-Code für Zugriff auf ein Arrayelement

Der **Zugriff auf ein Arrayelement** `ar[0]` wird im **Abstract Syntx Tree** in Code 0.14 mithilfe des **Container-Knotens** `Subscr(Name('ar'), Num('0'))` dargestellt.

```
1  File
2    Name './example_array_access.ast',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Assign(Alloc(Writeable(), ArrayDecl([Num('1')], IntType('int')), Name('ar')),
           ↪  Array([Num('42')]))
10         Exp(Subscr(Name('ar'), Num('0')))
11       ],
12     FunDef
13       VoidType 'void',
14       Name 'fun',
15       [],
16       [
17         Assign(Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')), Name('ar')),
           ↪  Array([Num('1'), Num('2'), Num('3')]))
18         Exp(Subscr(Name('ar'), BinOp(Num('1'), Add('+'), Num('1'))))
19       ]
20   ]
```

Code 0.14: Abstract Syntax Tree für Zugriff auf ein Arrayelement

Im **PicoC-Mon Pass** in Code 0.15 wird beim **Container-Knoten** `Subscr(Name('ar'), Num('0'))` zuerst die **Adresse** der Variable `Name('ar')` auf den **Stack** geschrieben. Bei den **Globalen Statischen Daten** der `main`-Funktion wird das durch die Komposition `Ref(Global(Num('0')))` dargestellt und beim **Stackframe** der Funktionm `fun` wird das durch die Komposition `Ref(Stackframe(Num('2')))` dargestellt.

In nächsten Schritt wird die Adresse des **Index**, des Arrays auf das Zugegriffen werden soll berechnet. Da der **Index** auf den Zugegriffen werden soll auch durch das Ergebnis eines **komplexeren Ausdrucks**, z.B. `<ar>[1 + <var2>]` bestimmt sein kann, indem auch **Variablen** vorkommen können, kann dieser nicht während des **Kompilierens** berechnet werden, sondern muss zur **Laufzeit** berechnet werden. Daher muss zuerst der Wert des **Index** dessen Adresse berechnet werden soll bestimmt werden, z.B. im einfachen Fall durch `Exp(Num('0'))` und dann muss der Index berechnet werden, was durch die Komposition `Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))` dargestellt wird.

```
File
  Name './example_array_access.picoc_mon',
  [
    Block
      Name 'main.1',
      [
        // Assign(Name('ar'), Array([Num('42')]))
        Exp(Num('42'))
        Assign(Global(Num('0')), Stack(Num('1')))
        // Exp(Subscr(Name('ar'), Num('0')))
        Ref(Global(Num('0')))
        Exp(Num('0'))
        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
        Exp(Stack(Num('1')))
        Return(Empty())
      ],
    Block
      Name 'fun.0',
      [
        // Assign(Name('ar'), Array([Num('1'), Num('2'), Num('3')]))
        Exp(Num('1'))
        Exp(Num('2'))
        Exp(Num('3'))
        Assign(Stackframe(Num('2')), Stack(Num('3')))
        // Exp(Subscr(Name('ar'), BinOp(Num('1'), Add('+'), Num('1'))))
        Ref(Stackframe(Num('2')))
        Exp(Num('1'))
        Exp(Num('1'))
        Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
        Exp(Stack(Num('1')))
        Return(Empty())
      ]
  ]
```

Code 0.15: PicoC-Mon Pass für Zugriff auf ein Arrayelement

```
File
  Name './example_array_access.reti_blocks',
  [
    Block
      Name 'main.1',
      [
        # // Assign(Name('ar'), Array([Num('42')]))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Assign(Global(Num('0')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 0;
        ADDI SP 1;
        # // Exp(Subscr(Name('ar'), Num('0')))
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
        # Exp(Num('0'))
        SUBI SP 1;
        LOADI ACC 0;
        STOREIN SP ACC 1;
        # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
        LOADIN SP IN1 2;
        LOADIN SP IN2 1;
        MULTI IN2 1;
        ADD IN1 IN2;
        ADDI SP 1;
        STOREIN SP IN1 1;
        # Exp(Stack(Num('1')))
        LOADIN SP IN1 1;
        LOADIN IN1 ACC 0;
        STOREIN SP ACC 1;
        # Return(Empty())
        LOADIN BAF PC -1;
      ],
    Block
      Name 'fun.0',
      [
        # // Assign(Name('ar'), Array([Num('1'), Num('2'), Num('3')]))
        # Exp(Num('1'))
        SUBI SP 1;
        LOADI ACC 1;
        STOREIN SP ACC 1;
        # Exp(Num('2'))
        SUBI SP 1;
        LOADI ACC 2;
        STOREIN SP ACC 1;
        # Exp(Num('3'))
        SUBI SP 1;
        LOADI ACC 3;
        STOREIN SP ACC 1;
        # Assign(Stackframe(Num('2')), Stack(Num('3')))
        LOADIN SP ACC 1;
```

```
58          STOREIN BAF ACC -2;
59          LOADIN SP ACC 2;
60          STOREIN BAF ACC -3;
61          LOADIN SP ACC 3;
62          STOREIN BAF ACC -4;
63          ADDI SP 3;
64          # // Exp(Subscr(Name('ar'), BinOp(Num('1'), Add('+'), Num('1'))))
65          # Ref(Stackframe(Num('2')))
66          SUBI SP 1;
67          MOVE BAF IN1;
68          SUBI IN1 4;
69          STOREIN SP IN1 1;
70          # Exp(Num('1'))
71          SUBI SP 1;
72          LOADI ACC 1;
73          STOREIN SP ACC 1;
74          # Exp(Num('1'))
75          SUBI SP 1;
76          LOADI ACC 1;
77          STOREIN SP ACC 1;
78          # Exp(BinOp(Stack(Num('2')), Add('+'), Stack(Num('1'))))
79          LOADIN SP ACC 2;
80          LOADIN SP IN2 1;
81          ADD ACC IN2;
82          STOREIN SP ACC 2;
83          ADDI SP 1;
84          # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
85          LOADIN SP IN1 2;
86          LOADIN SP IN2 1;
87          MULTI IN2 1;
88          ADD IN1 IN2;
89          ADDI SP 1;
90          STOREIN SP IN1 1;
91          # Exp(Stack(Num('1')))
92          LOADIN SP IN1 1;
93          LOADIN IN1 ACC 0;
94          STOREIN SP ACC 1;
95          # Return(Empty())
96          LOADIN BAF PC -1;
97      ]
98  ]
```

Code 0.16: RETI-Blocks Pass für Zugriff auf ein Arrayelement

0.0.2.3 Zuweisung an Arrayindex

```
1 void main() {
2   int ar[2];
3   ar[2] = 42;
4 }
```

Code 0.17: PicoC Code für Zuweisung an Arrayindex

```
File
  Name './example_array_assignment.ast',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Exp(Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar')))
        Assign(Subscr(Name('ar'), Num('2')), Num('42'))
      ]
  ]
```

Code 0.18: Abstract Syntax Tree für Zuweisung an Arrayindex

```
File
  Name './example_array_assignment.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Exp(Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar')))
        // Assign(Subscr(Name('ar'), Num('2')), Num('42'))
        Exp(Num('42'))
        Ref(Global(Num('0')))
        Exp(Num('2'))
        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
        Assign(Stack(Num('1')), Stack(Num('2')))
        Return(Empty())
      ]
  ]
```

Code 0.19: PicoC Mon Pass für Zuweisung an Arrayindex

```
File
  Name './example_array_assignment.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Exp(Alloc(Writeable(), ArrayDecl([Num('2')], IntType('int')), Name('ar')))
        # // Assign(Subscr(Name('ar'), Num('2')), Num('42'))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
```

```
18        # Exp(Num('2'))
19        SUBI SP 1;
20        LOADI ACC 2;
21        STOREIN SP ACC 1;
22        # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
23        LOADIN SP IN1 2;
24        LOADIN SP IN2 1;
25        MULTI IN2 1;
26        ADD IN1 IN2;
27        ADDI SP 1;
28        STOREIN SP IN1 1;
29        LOADIN SP IN1 1;
30        LOADIN SP ACC 2;
31        ADDI SP 2;
32        STOREIN IN1 ACC 0;
33        # Return(Empty())
34        LOADIN BAF PC -1;
35      ]
36  ]
```

Code 0.20: RETI Blocks Pass für Zuweisung an Arrayindex

0.0.3 Umsetzung von Structs

0.0.3.1 Deklaration von Structs

```
1 struct st1 {int *ar[3];};
2
3 struct st2 {struct st1 st;};
4
5 void main() {
6 }
```

Code 0.21: PicoC Code für Deklaration von Structs

```
1  SymbolTable
2    [
3      Symbol
4        {
5          type qualifier:         Empty()
6          datatype:               ArrayDecl([Num('3')], PntrDecl(Num('1'), IntType('int')))
7          name:                   Name('ar@st1')
8          value or address:       Empty()
9          position:               Pos(Num('1'), Num('17'))
10         size:                   Num('3')
11       },
12     Symbol
13       {
14         type qualifier:         Empty()
15         datatype:               StructDecl(Name('st1'), [Alloc(Writeable(),
           ↪  ArrayDecl([Num('3')], PntrDecl(Num('1'), IntType('int'))), Name('ar'))])
```

```
16        name:                   Name('st1')
17        value or address:       [Name('ar@st1')]
18        position:               Pos(Num('1'), Num('7'))
19        size:                   Num('3')
20      },
21    Symbol
22      {
23        type qualifier:         Empty()
24        datatype:               StructSpec(Name('st1'))
25        name:                   Name('st@st2')
26        value or address:       Empty()
27        position:               Pos(Num('3'), Num('23'))
28        size:                   Num('3')
29      },
30    Symbol
31      {
32        type qualifier:         Empty()
33        datatype:               StructDecl(Name('st2'), [Alloc(Writeable(),
          ↪  StructSpec(Name('st1')), Name('st'))])
34        name:                   Name('st2')
35        value or address:       [Name('st@st2')]
36        position:               Pos(Num('3'), Num('7'))
37        size:                   Num('3')
38      },
39    Symbol
40      {
41        type qualifier:         Empty()
42        datatype:               FunDecl(VoidType('void'), Name('main'), [])
43        name:                   Name('main')
44        value or address:       Empty()
45        position:               Pos(Num('5'), Num('5'))
46        size:                   Empty()
47      }
48  ]
```

Code 0.22: Symboltabelle für Deklaration von Structs

0.0.3.2 Initialisierung von Structs

```
1 struct st1 {int *pntr[1];};
2
3 struct st2 {struct st1 st;};
4
5 void main() {
6   int var = 42;
7   struct st1 st = {.st={.pntr={{&var}}}};
8 }
```

Code 0.23: PicoC Code für Initialisierung von Structs

```
1 File
2   Name './example_struct_init.ast',
```

```
  [
    StructDecl
      Name 'st1',
      [
        Alloc
          Writeable,
          ArrayDecl
            [
              Num '1'
            ],
            PntrDecl
              Num '1',
              IntType 'int',
          Name 'pntr'
      ],
    StructDecl
      Name 'st2',
      [
        Alloc
          Writeable,
          StructSpec
            Name 'st1',
          Name 'st'
      ],
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
        Assign(Alloc(Writeable(), StructSpec(Name('st1')), Name('st')),
        ↪  Struct([Assign(Name('st'), Struct([Assign(Name('pntr'),
        ↪  Array([Array([Ref(Name('var'))])])])])]))
      ]
  ]
```

Code 0.24: Abstract Syntax Tree für Initialisierung von Structs

```
SymbolTable
  [
    Symbol
      {
        type qualifier:         Empty()
        datatype:               ArrayDecl([Num('1')], PntrDecl(Num('1'), IntType('int')))
        name:                   Name('pntr@st1')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('17'))
        size:                   Num('1')
      },
    Symbol
      {
        type qualifier:         Empty()
        datatype:               StructDecl(Name('st1'), [Alloc(Writeable(),
        ↪  ArrayDecl([Num('1')], PntrDecl(Num('1'), IntType('int'))), Name('pntr'))])
```

```
16        name:                   Name('st1')
17        value or address:       [Name('pntr@st1')]
18        position:               Pos(Num('1'), Num('7'))
19        size:                   Num('1')
20      },
21    Symbol
22      {
23        type qualifier:         Empty()
24        datatype:               StructSpec(Name('st1'))
25        name:                   Name('st@st2')
26        value or address:       Empty()
27        position:               Pos(Num('3'), Num('23'))
28        size:                   Num('1')
29      },
30    Symbol
31      {
32        type qualifier:         Empty()
33        datatype:               StructDecl(Name('st2'), [Alloc(Writeable(),
          ↪  StructSpec(Name('st1')), Name('st'))])
34        name:                   Name('st2')
35        value or address:       [Name('st@st2')]
36        position:               Pos(Num('3'), Num('7'))
37        size:                   Num('1')
38      },
39    Symbol
40      {
41        type qualifier:         Empty()
42        datatype:               FunDecl(VoidType('void'), Name('main'), [])
43        name:                   Name('main')
44        value or address:       Empty()
45        position:               Pos(Num('5'), Num('5'))
46        size:                   Empty()
47      },
48    Symbol
49      {
50        type qualifier:         Writeable()
51        datatype:               IntType('int')
52        name:                   Name('var@main')
53        value or address:       Num('0')
54        position:               Pos(Num('6'), Num('6'))
55        size:                   Num('1')
56      },
57    Symbol
58      {
59        type qualifier:         Writeable()
60        datatype:               StructSpec(Name('st1'))
61        name:                   Name('st@main')
62        value or address:       Num('1')
63        position:               Pos(Num('7'), Num('13'))
64        size:                   Num('1')
65      }
66  ]
```

Code 0.25: Symboltabelle für Initialisierung von Structs

```
File
  Name './example_struct_init.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('var'), Num('42'))
        Exp(Num('42'))
        Assign(Global(Num('0')), Stack(Num('1')))
        // Assign(Name('st'), Struct([Assign(Name('st'), Struct([Assign(Name('pntr'),
        ↪  Array([Array([Ref(Name('var'))])]))])]))
        Ref(Global(Num('0')))
        Assign(Global(Num('1')), Stack(Num('1')))
        Return(Empty())
      ]
  ]
```

Code 0.26: PicoC Mon Pass für Initialisierung von Structs

```
File
  Name './example_struct_init.reti_blocks',
  [
    Block
      Name 'main.0',
      [
        # // Assign(Name('var'), Num('42'))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Assign(Global(Num('0')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 0;
        ADDI SP 1;
        # // Assign(Name('st'), Struct([Assign(Name('st'), Struct([Assign(Name('pntr'),
        ↪  Array([Array([Ref(Name('var'))])]))])]))
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
        # Assign(Global(Num('1')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 1;
        ADDI SP 1;
        # Return(Empty())
        LOADIN BAF PC -1;
      ]
  ]
```

Code 0.27: RETI Blocks Pass für Initialisierung von Structs

0.0.3.3 Zugriff auf Structattribut

```
struct pos {int x; int y;};

void main() {
  struct pos st = {.x=4, .y=2};
  st.y;
}
```

Code 0.28: PicoC Code für Zugriff auf Structattribut

```
File
  Name './example_struct_attr_access.ast',
  [
    StructDecl
      Name 'pos',
      [
        Alloc
          Writeable,
          IntType 'int',
          Name 'x',
        Alloc
          Writeable,
          IntType 'int',
          Name 'y'
      ],
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Assign(Alloc(Writeable(), StructSpec(Name('pos')), Name('st')),
        ↪  Struct([Assign(Name('x'), Num('4')), Assign(Name('y'), Num('2'))]))
        Exp(Attr(Name('st'), Name('y')))
      ]
  ]
```

Code 0.29: Abstract Syntax Tree für Zugriff auf Structattribut

```
File
  Name './example_struct_attr_access.picoc_mon',
  [
    Block
      Name 'main.0',
      [
        // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
        ↪  Num('2'))]))
        Exp(Num('4'))
        Exp(Num('2'))
        Assign(Global(Num('0')), Stack(Num('2')))
        // Exp(Attr(Name('st'), Name('y')))
```

```
12        Ref(Global(Num('0')))
13        Ref(Attr(Stack(Num('1')), Name('y')))
14        Exp(Stack(Num('1')))
15        Return(Empty())
16      ]
17  ]
```

Code 0.30: PicoC Mon Pass für Zugriff auf Structattribut

```
1 File
2   Name './example_struct_attr_access.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
          ↪  Num('2'))]))
8         # Exp(Num('4'))
9         SUBI SP 1;
10        LOADI ACC 4;
11        STOREIN SP ACC 1;
12        # Exp(Num('2'))
13        SUBI SP 1;
14        LOADI ACC 2;
15        STOREIN SP ACC 1;
16        # Assign(Global(Num('0')), Stack(Num('2')))
17        LOADIN SP ACC 1;
18        STOREIN DS ACC 1;
19        LOADIN SP ACC 2;
20        STOREIN DS ACC 0;
21        ADDI SP 2;
22        # // Exp(Attr(Name('st'), Name('y')))
23        # Ref(Global(Num('0')))
24        SUBI SP 1;
25        LOADI IN1 0;
26        ADD IN1 DS;
27        STOREIN SP IN1 1;
28        # Ref(Attr(Stack(Num('1')), Name('y')))
29        LOADIN SP IN1 1;
30        ADDI IN1 1;
31        STOREIN SP IN1 1;
32        # Exp(Stack(Num('1')))
33        LOADIN SP IN1 1;
34        LOADIN IN1 ACC 0;
35        STOREIN SP ACC 1;
36        # Return(Empty())
37        LOADIN BAF PC -1;
38      ]
39  ]
```

Code 0.31: RETI Blocks Pass für Zugriff auf Structattribut

0.0.3.4 Zuweisung an Structattribut

```
1 struct pos {int x; int y;};
2
3 void main() {
4   struct pos st = {.x=4, .y=2};
5   st.y = 42;
6 }
```

Code 0.32: PicoC Code für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.ast',
3   [
4     StructDecl
5       Name 'pos',
6       [
7         Alloc
8           Writeable,
9           IntType 'int',
10          Name 'x',
11        Alloc
12          Writeable,
13          IntType 'int',
14          Name 'y'
15      ],
16    FunDef
17      VoidType 'void',
18      Name 'main',
19      [],
20      [
21        Assign(Alloc(Writeable(), StructSpec(Name('pos')), Name('st')),
          ↪  Struct([Assign(Name('x'), Num('4')), Assign(Name('y'), Num('2'))]))
22        Assign(Attr(Name('st'), Name('y')), Num('42'))
23      ]
24  ]
```

Code 0.33: Abstract Syntax Tree für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
          ↪  Num('2'))]))
8         Exp(Num('4'))
9         Exp(Num('2'))
10        Assign(Global(Num('0')), Stack(Num('2')))
11        // Assign(Attr(Name('st'), Name('y')), Num('42'))
```

```
12        Exp(Num('42'))
13        Ref(Global(Num('0')))
14        Ref(Attr(Stack(Num('1')), Name('y')))
15        Assign(Stack(Num('1')), Stack(Num('2')))
16        Return(Empty())
17      ]
18    ]
```

Code 0.34: PicoC Mon Pass für Zuweisung an Structattribut

```
1 File
2   Name './example_struct_attr_assignment.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('st'), Struct([Assign(Name('x'), Num('4')), Assign(Name('y'),
          ↪  Num('2'))]))
8         # Exp(Num('4'))
9         SUBI SP 1;
10        LOADI ACC 4;
11        STOREIN SP ACC 1;
12        # Exp(Num('2'))
13        SUBI SP 1;
14        LOADI ACC 2;
15        STOREIN SP ACC 1;
16        # Assign(Global(Num('0')), Stack(Num('2')))
17        LOADIN SP ACC 1;
18        STOREIN DS ACC 1;
19        LOADIN SP ACC 2;
20        STOREIN DS ACC 0;
21        ADDI SP 2;
22        # // Assign(Attr(Name('st'), Name('y')), Num('42'))
23        # Exp(Num('42'))
24        SUBI SP 1;
25        LOADI ACC 42;
26        STOREIN SP ACC 1;
27        # Ref(Global(Num('0')))
28        SUBI SP 1;
29        LOADI IN1 0;
30        ADD IN1 DS;
31        STOREIN SP IN1 1;
32        # Ref(Attr(Stack(Num('1')), Name('y')))
33        LOADIN SP IN1 1;
34        ADDI IN1 1;
35        STOREIN SP IN1 1;
36        LOADIN SP IN1 1;
37        LOADIN SP ACC 2;
38        ADDI SP 2;
39        STOREIN IN1 ACC 0;
40        # Return(Empty())
41        LOADIN BAF PC -1;
42      ]
43    ]
```

Code 0.35: RETI Blocks Pass für Zuweisung an Structattribut

0.0.4 Umsetzung der Derived Datatypes im Zusammenspiel

0.0.4.1 Einleitungsteil für Globale Statische Daten und Stackframe

```
1  struct ar_with_len {int len; int ar[2];};
2
3  void main() {
4    struct ar_with_len st_ar[3];
5    int *(*pntr2)[3];
6    pntr2;
7  }
8
9  void fun() {
10   struct ar_with_len st_ar[3];
11   int (*pntr1)[3];
12   pntr1;
13 }
```

Code 0.36: PicoC Code für den Einleitungsteil

```
1  File
2    Name './example_derived_dts_introduction_part.ast',
3    [
4      StructDecl
5        Name 'ar_with_len',
6        [
7          Alloc
8            Writeable,
9            IntType 'int',
10           Name 'len',
11         Alloc
12           Writeable,
13           ArrayDecl
14             [
15               Num '2'
16             ],
17             IntType 'int',
18           Name 'ar'
19       ],
20     FunDef
21       VoidType 'void',
22       Name 'main',
23       [],
24       [
25         Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
           ↪  Name('st_ar')))
26         Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], PntrDecl(Num('1'),
           ↪  IntType('int')))), Name('pntr2')))
27         Exp(Name('pntr2'))
```

```
28      ],
29    FunDef
30      VoidType 'void',
31      Name 'fun',
32      [],
33      [
34        Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
35        Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
          ↪  Name('pntr1')))
36        Exp(Name('pntr1'))
37      ]
38  ]
```

Code 0.37: Abstract Syntax Tree für den Einleitungsteil

```
1 File
2   Name './example_derived_dts_introduction_part.picoc_mon',
3   [
4     Block
5       Name 'main.1',
6       [
7         // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
8         // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], PntrDecl(Num('1'),
          ↪  IntType('int')))), Name('pntr2')))
9         // Exp(Name('pntr2'))
10        Exp(Global(Num('9')))
11        Return(Empty())
12      ],
13    Block
14      Name 'fun.0',
15      [
16        // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
17        // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
          ↪  Name('pntr1')))
18        // Exp(Name('pntr1'))
19        Exp(Stackframe(Num('9')))
20        Return(Empty())
21      ]
22  ]
```

Code 0.38: PicoC Mon Pass für den Einleitungsteil

```
1 File
2   Name './example_derived_dts_introduction_part.reti_blocks',
3   [
4     Block
5       Name 'main.1',
6       [
```

```
7         # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
8         # // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')],
          ↪  PntrDecl(Num('1'), IntType('int')))), Name('pntr2')))
9         # // Exp(Name('pntr2'))
10        # Exp(Global(Num('9')))
11        SUBI SP 1;
12        LOADIN DS ACC 9;
13        STOREIN SP ACC 1;
14        # Return(Empty())
15        LOADIN BAF PC -1;
16      ],
17    Block
18      Name 'fun.0',
19      [
20        # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('ar_with_len'))),
          ↪  Name('st_ar')))
21        # // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')],
          ↪  IntType('int'))), Name('pntr1')))
22        # // Exp(Name('pntr1'))
23        # Exp(Stackframe(Num('9')))
24        SUBI SP 1;
25        LOADIN BAF ACC -11;
26        STOREIN SP ACC 1;
27        # Return(Empty())
28        LOADIN BAF PC -1;
29      ]
30  ]
```

Code 0.39: RETI Blocks Pass für den Einleitungsteil

0.0.4.2 Mittelteil für die verschiedenen Derived Datatypes

```
1 struct st1 {int (*ar)[1];};
2
3 void main() {
4   int var[1] = {42};
5   struct st1 st_first = {.ar=&var};
6   (*st_first.ar)[0];
7 }
```

Code 0.40: PicoC Code für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.ast',
3   [
4     StructDecl
5       Name 'st1',
6       [
7         Alloc
8           Writeable,
9           PntrDecl
```

```
10            Num '1',
11            ArrayDecl
12              [
13                Num '1'
14              ],
15              IntType 'int',
16          Name 'ar'
17      ],
18    FunDef
19      VoidType 'void',
20      Name 'main',
21      [],
22      [
23        Assign(Alloc(Writeable(), ArrayDecl([Num('1')], IntType('int')), Name('var')),
          ↪  Array([Num('42')]))
24        Assign(Alloc(Writeable(), StructSpec(Name('st1')), Name('st_first')),
          ↪  Struct([Assign(Name('ar'), Ref(Name('var')))]))
25        Exp(Subscr(Deref(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0')))
26      ]
27  ]
```

Code 0.41: Abstract Syntax Tree für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('var'), Array([Num('42')]))
8         Exp(Num('42'))
9         Assign(Global(Num('0')), Stack(Num('1')))
10        // Assign(Name('st_first'), Struct([Assign(Name('ar'), Ref(Name('var')))]))
11        Ref(Global(Num('0')))
12        Assign(Global(Num('1')), Stack(Num('1')))
13        // Exp(Subscr(Subscr(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0')))
14        Ref(Global(Num('1')))
15        Ref(Attr(Stack(Num('1')), Name('ar')))
16        Exp(Num('0'))
17        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
18        Exp(Num('0'))
19        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
20        Exp(Stack(Num('1')))
21        Return(Empty())
22      ]
23  ]
```

Code 0.42: PicoC Mon Pass für den Mittelteil

```
1 File
2   Name './example_derived_dts_main_part.reti_blocks',
```

```
3  [
4    Block
5      Name 'main.0',
6      [
7        # // Assign(Name('var'), Array([Num('42')]))
8        # Exp(Num('42'))
9        SUBI SP 1;
10       LOADI ACC 42;
11       STOREIN SP ACC 1;
12       # Assign(Global(Num('0')), Stack(Num('1')))
13       LOADIN SP ACC 1;
14       STOREIN DS ACC 0;
15       ADDI SP 1;
16       # // Assign(Name('st_first'), Struct([Assign(Name('ar'), Ref(Name('var')))]))
17       # Ref(Global(Num('0')))
18       SUBI SP 1;
19       LOADI IN1 0;
20       ADD IN1 DS;
21       STOREIN SP IN1 1;
22       # Assign(Global(Num('1')), Stack(Num('1')))
23       LOADIN SP ACC 1;
24       STOREIN DS ACC 1;
25       ADDI SP 1;
26       # // Exp(Subscr(Subscr(Attr(Name('st_first'), Name('ar')), Num('0')), Num('0')))
27       # Ref(Global(Num('1')))
28       SUBI SP 1;
29       LOADI IN1 1;
30       ADD IN1 DS;
31       STOREIN SP IN1 1;
32       # Ref(Attr(Stack(Num('1')), Name('ar')))
33       LOADIN SP IN1 1;
34       ADDI IN1 0;
35       STOREIN SP IN1 1;
36       # Exp(Num('0'))
37       SUBI SP 1;
38       LOADI ACC 0;
39       STOREIN SP ACC 1;
40       # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
41       LOADIN SP IN2 2;
42       LOADIN IN2 IN1 0;
43       LOADIN SP IN2 1;
44       MULTI IN2 1;
45       ADD IN1 IN2;
46       ADDI SP 1;
47       STOREIN SP IN1 1;
48       # Exp(Num('0'))
49       SUBI SP 1;
50       LOADI ACC 0;
51       STOREIN SP ACC 1;
52       # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
53       LOADIN SP IN1 2;
54       LOADIN SP IN2 1;
55       MULTI IN2 1;
56       ADD IN1 IN2;
57       ADDI SP 1;
58       STOREIN SP IN1 1;
59       # Exp(Stack(Num('1')))
```

```
60          LOADIN SP IN1 1;
61          LOADIN IN1 ACC 0;
62          STOREIN SP ACC 1;
63          # Return(Empty())
64          LOADIN BAF PC -1;
65        ]
66    ]
```

Code 0.43: RETI Blocks Pass für den Mittelteil

0.0.4.3 Schlussteil für die verschiedenen Derived Datatypes

```
1  struct st {int attr[2];};
2
3  void main() {
4    int ar1[1][2] = {{42, 314}};
5    struct st ar2[1] = {.attr={42, 314}};
6    int var = 42;
7    int *pntr1 = &var;
8    int **pntr2 = &pntr1;
9
10   ar1[0];
11   ar2[0];
12   *pntr2;
13 }
```

Code 0.44: PicoC Code für den Schlussteil

```
1  File
2    Name './example_derived_dts_final_part.ast',
3    [
4      StructDecl
5        Name 'st',
6        [
7          Alloc
8            Writeable,
9            ArrayDecl
10             [
11               Num '2'
12             ],
13             IntType 'int',
14           Name 'attr'
15       ],
16     FunDef
17       VoidType 'void',
18       Name 'main',
19       [],
20       [
21         Assign(Alloc(Writeable(), ArrayDecl([Num('1'), Num('2')], IntType('int')),
           ↪  Name('ar1')), Array([Array([Num('42'), Num('314')])]))
22         Assign(Alloc(Writeable(), ArrayDecl([Num('1')], StructSpec(Name('st'))),
           ↪  Name('ar2')), Struct([Assign(Name('attr'), Array([Num('42'), Num('314')]))]))
```

```
23        Assign(Alloc(Writeable(), IntType('int'), Name('var')), Num('42'))
24        Assign(Alloc(Writeable(), PntrDecl(Num('1'), IntType('int')), Name('pntr1')),
          ↪  Ref(Name('var')))
25        Assign(Alloc(Writeable(), PntrDecl(Num('2'), IntType('int')), Name('pntr2')),
          ↪  Ref(Name('pntr1')))
26        Exp(Subscr(Name('ar1'), Num('0')))
27        Exp(Subscr(Name('ar2'), Num('0')))
28        Exp(Deref(Name('pntr2'), Num('0')))
29      ]
30    ]
```

Code 0.45: Abstract Syntax Tree für den Schlussteil

```
1 File
2   Name './example_derived_dts_final_part.picoc_mon',
3   [
4     Block
5       Name 'main.0',
6       [
7         // Assign(Name('ar1'), Array([Array([Num('42'), Num('314')])]))
8         Exp(Num('42'))
9         Exp(Num('314'))
10        Assign(Global(Num('0')), Stack(Num('2')))
11        // Assign(Name('ar2'), Struct([Assign(Name('attr'), Array([Num('42'),
          ↪  Num('314')]))]))
12        Exp(Num('42'))
13        Exp(Num('314'))
14        Assign(Global(Num('2')), Stack(Num('2')))
15        // Assign(Name('var'), Num('42'))
16        Exp(Num('42'))
17        Assign(Global(Num('4')), Stack(Num('1')))
18        // Assign(Name('pntr1'), Ref(Name('var')))
19        Ref(Global(Num('4')))
20        Assign(Global(Num('5')), Stack(Num('1')))
21        // Assign(Name('pntr2'), Ref(Name('pntr1')))
22        Ref(Global(Num('5')))
23        Assign(Global(Num('6')), Stack(Num('1')))
24        // Exp(Subscr(Name('ar1'), Num('0')))
25        Ref(Global(Num('0')))
26        Exp(Num('0'))
27        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
28        Exp(Stack(Num('1')))
29        // Exp(Subscr(Name('ar2'), Num('0')))
30        Ref(Global(Num('2')))
31        Exp(Num('0'))
32        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
33        Exp(Stack(Num('1')))
34        // Exp(Subscr(Name('pntr2'), Num('0')))
35        Ref(Global(Num('6')))
36        Exp(Num('0'))
37        Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
38        Exp(Stack(Num('1')))
39        Return(Empty())
40      ]
```

```
41  ]
```

Code 0.46: PicoC Mon Pass für den Schlussteil

```
1 File
2   Name './example_derived_dts_final_part.reti_blocks',
3   [
4     Block
5       Name 'main.0',
6       [
7         # // Assign(Name('ar1'), Array([Array([Num('42'), Num('314')])]))
8         # Exp(Num('42'))
9         SUBI SP 1;
10        LOADI ACC 42;
11        STOREIN SP ACC 1;
12        # Exp(Num('314'))
13        SUBI SP 1;
14        LOADI ACC 314;
15        STOREIN SP ACC 1;
16        # Assign(Global(Num('0')), Stack(Num('2')))
17        LOADIN SP ACC 1;
18        STOREIN DS ACC 1;
19        LOADIN SP ACC 2;
20        STOREIN DS ACC 0;
21        ADDI SP 2;
22        # // Assign(Name('ar2'), Struct([Assign(Name('attr'), Array([Num('42'),
          ↪  Num('314')]))]))
23        # Exp(Num('42'))
24        SUBI SP 1;
25        LOADI ACC 42;
26        STOREIN SP ACC 1;
27        # Exp(Num('314'))
28        SUBI SP 1;
29        LOADI ACC 314;
30        STOREIN SP ACC 1;
31        # Assign(Global(Num('2')), Stack(Num('2')))
32        LOADIN SP ACC 1;
33        STOREIN DS ACC 3;
34        LOADIN SP ACC 2;
35        STOREIN DS ACC 2;
36        ADDI SP 2;
37        # // Assign(Name('var'), Num('42'))
38        # Exp(Num('42'))
39        SUBI SP 1;
40        LOADI ACC 42;
41        STOREIN SP ACC 1;
42        # Assign(Global(Num('4')), Stack(Num('1')))
43        LOADIN SP ACC 1;
44        STOREIN DS ACC 4;
45        ADDI SP 1;
46        # // Assign(Name('pntr1'), Ref(Name('var')))
47        # Ref(Global(Num('4')))
48        SUBI SP 1;
49        LOADI IN1 4;
```

```
50          ADD IN1 DS;
51          STOREIN SP IN1 1;
52          # Assign(Global(Num('5')), Stack(Num('1')))
53          LOADIN SP ACC 1;
54          STOREIN DS ACC 5;
55          ADDI SP 1;
56          # // Assign(Name('pntr2'), Ref(Name('pntr1')))
57          # Ref(Global(Num('5')))
58          SUBI SP 1;
59          LOADI IN1 5;
60          ADD IN1 DS;
61          STOREIN SP IN1 1;
62          # Assign(Global(Num('6')), Stack(Num('1')))
63          LOADIN SP ACC 1;
64          STOREIN DS ACC 6;
65          ADDI SP 1;
66          # // Exp(Subscr(Name('ar1'), Num('0')))
67          # Ref(Global(Num('0')))
68          SUBI SP 1;
69          LOADI IN1 0;
70          ADD IN1 DS;
71          STOREIN SP IN1 1;
72          # Exp(Num('0'))
73          SUBI SP 1;
74          LOADI ACC 0;
75          STOREIN SP ACC 1;
76          # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
77          LOADIN SP IN1 2;
78          LOADIN SP IN2 1;
79          MULTI IN2 2;
80          ADD IN1 IN2;
81          ADDI SP 1;
82          STOREIN SP IN1 1;
83          # Exp(Stack(Num('1')))
84          # // Exp(Subscr(Name('ar2'), Num('0')))
85          # Ref(Global(Num('2')))
86          SUBI SP 1;
87          LOADI IN1 2;
88          ADD IN1 DS;
89          STOREIN SP IN1 1;
90          # Exp(Num('0'))
91          SUBI SP 1;
92          LOADI ACC 0;
93          STOREIN SP ACC 1;
94          # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
95          LOADIN SP IN1 2;
96          LOADIN SP IN2 1;
97          MULTI IN2 2;
98          ADD IN1 IN2;
99          ADDI SP 1;
100         STOREIN SP IN1 1;
101         # Exp(Stack(Num('1')))
102         LOADIN SP IN1 1;
103         LOADIN IN1 ACC 0;
104         STOREIN SP ACC 1;
105         # // Exp(Subscr(Name('pntr2'), Num('0')))
106         # Ref(Global(Num('6')))
```

```
107         SUBI SP 1;
108         LOADI IN1 6;
109         ADD IN1 DS;
110         STOREIN SP IN1 1;
111         # Exp(Num('0'))
112         SUBI SP 1;
113         LOADI ACC 0;
114         STOREIN SP ACC 1;
115         # Ref(Subscr(Stack(Num('2')), Stack(Num('1'))))
116         LOADIN SP IN2 2;
117         LOADIN IN2 IN1 0;
118         LOADIN SP IN2 1;
119         MULTI IN2 1;
120         ADD IN1 IN2;
121         ADDI SP 1;
122         STOREIN SP IN1 1;
123         # Exp(Stack(Num('1')))
124         # Return(Empty())
125         LOADIN BAF PC -1;
126       ]
127   ]
```

Code 0.47: RETI Blocks Pass für den Schlussteil

0.0.5 Umsetzung von Funktionen

0.0.5.1 Funktionen auflösen zu RETI Code

```
1  void main() {
2    return;
3  }
4
5  void fun1() {
6  }
7
8  int fun2() {
9    return 1;
10 }
```

Code 0.48: PicoC Code für 3 Funktionen

```
1  File
2    Name './example_3_funs.ast',
3    [
4      FunDef
5        VoidType 'void',
6        Name 'main',
7        [],
8        [
9          Return(Empty())
10       ],
```

```
    FunDef
      VoidType 'void',
      Name 'fun1',
      [],
      [],
    FunDef
      IntType 'int',
      Name 'fun2',
      [],
      [
        Return(Num('1'))
      ]
  ]
```

Code 0.49: Abstract Syntax Tree für 3 Funktionen

```
File
  Name './example_3_funs.picoc_blocks',
  [
    FunDef
      VoidType 'void',
      Name 'main',
      [],
      [
        Block
          Name 'main.2',
          [
            Return(Empty())
          ]
      ],
    FunDef
      VoidType 'void',
      Name 'fun1',
      [],
      [
        Block
          Name 'fun1.1',
          []
      ],
    FunDef
      IntType 'int',
      Name 'fun2',
      [],
      [
        Block
          Name 'fun2.0',
          [
            Return(Num('1'))
          ]
      ]
  ]
```

Code 0.50: PicoC Blocks Pass für 3 Funktionen

```
File
  Name './example_3_funs.picoc_mon',
  [
    Block
      Name 'main.2',
      [
        Return(Empty())
      ],
    Block
      Name 'fun1.1',
      [
        Return(Empty())
      ],
    Block
      Name 'fun2.0',
      [
        // Return(Num('1'))
        Exp(Num('1'))
        Return(Stack(Num('1')))
      ]
  ]
```

Code 0.51: PicoC Mon Pass für 3 Funktionen

```
File
  Name './example_3_funs.reti_blocks',
  [
    Block
      Name 'main.2',
      [
        # Return(Empty())
        LOADIN BAF PC -1;
      ],
    Block
      Name 'fun1.1',
      [
        # Return(Empty())
        LOADIN BAF PC -1;
      ],
    Block
      Name 'fun2.0',
      [
        # // Return(Num('1'))
        # Exp(Num('1'))
        SUBI SP 1;
        LOADI ACC 1;
        STOREIN SP ACC 1;
        # Return(Stack(Num('1')))
        LOADIN SP ACC 1;
        ADDI SP 1;
        LOADIN BAF PC -1;
      ]
  ]
```

Code 0.52: RETI Blocks Pass für 3 Funktionen

0.0.5.1.1 Sprung zur Main Funktion

```
void fun1() {
}

int fun2() {
  return 1;
}

void main() {
  return;
}
```

Code 0.53: PicoC Code für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
File
  Name './example_3_funs_main.picoc_mon',
  [
    Block
      Name 'fun1.2',
      [
        Return(Empty())
      ],
    Block
      Name 'fun2.1',
      [
        // Return(Num('1'))
        Exp(Num('1'))
        Return(Stack(Num('1')))
      ],
    Block
      Name 'main.0',
      [
        Return(Empty())
      ]
  ]
```

Code 0.54: PicoC Mon Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
File
  Name './example_3_funs_main.reti_blocks',
  [
    Block
      Name 'fun1.2',
      [
        # Return(Empty())
```

```
        LOADIN BAF PC -1;
      ],
    Block
      Name 'fun2.1',
      [
        # // Return(Num('1'))
        # Exp(Num('1'))
        SUBI SP 1;
        LOADI ACC 1;
        STOREIN SP ACC 1;
        # Return(Stack(Num('1')))
        LOADIN SP ACC 1;
        ADDI SP 1;
        LOADIN BAF PC -1;
      ],
    Block
      Name 'main.0',
      [
        # Return(Empty())
        LOADIN BAF PC -1;
      ]
  ]
```

Code 0.55: PicoC Blocks Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

```
File
  Name './example_3_funs_main.reti_patch',
  [
    Block
      Name 'start.3',
      [
        # // Exp(GoTo(Name('main.0')))
        Exp(GoTo(Name('main.0')))
      ],
    Block
      Name 'fun1.2',
      [
        # Return(Empty())
        LOADIN BAF PC -1;
      ],
    Block
      Name 'fun2.1',
      [
        # // Return(Num('1'))
        # Exp(Num('1'))
        SUBI SP 1;
        LOADI ACC 1;
        STOREIN SP ACC 1;
        # Return(Stack(Num('1')))
        LOADIN SP ACC 1;
        ADDI SP 1;
        LOADIN BAF PC -1;
      ],
    Block
```

```
30      Name 'main.0',
31      [
32        # Return(Empty())
33        LOADIN BAF PC -1;
34      ]
35  ]
```

Code 0.56: PicoC Patch Pass für Funktionen, wobei die main Funktion nicht die erste Funktion ist

0.0.5.2 Funktionsdeklaration und -definition und Umsetzung von Scopes

```
1  int fun2(int var);
2
3  void fun1() {
4  }
5
6  void main() {
7    int var = fun2(42);
8    return;
9  }
10
11 int fun2(int var) {
12   return var;
13 }
```

Code 0.57: PicoC Code für Funktionen, wobei eine Funktion vorher deklariert werden muss

Bei mehreren Funktionen werden die **Scopes** der unterschiedlichen **Funktionen** mittels eines **Suffix** `"<fun_name>@"` umgesetzt, der an den **Variablennamen** `<var>` drangehängt wird: `<var>@<fun_name>`. Dieser **Suffix** wird geändert sobald beim **Top-Down**[4] Durchiterieren über den **Abstract Syntax Tree** des aktuellen **Passes** nach dem **Depth-First-Search** Schema über den

```
SymbolTable
  [
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(IntType('int'), Name('fun2'), [Alloc(Writeable(),
        ↪  IntType('int'), Name('var'))])
        name:                   Name('fun2')
        value or address:       Empty()
        position:               Pos(Num('1'), Num('4'))
        size:                   Empty()
      },
    Symbol
      {
        type qualifier:         Empty()
        datatype:               FunDecl(VoidType('void'), Name('fun1'), [])
        name:                   Name('fun1')
        value or address:       Empty()
```

[4] D.h. von der Wurzel zu den Blättern eines Baumes

```
18        position:               Pos(Num('3'), Num('5'))
19        size:                   Empty()
20      },
21    Symbol
22      {
23        type qualifier:         Empty()
24        datatype:               FunDecl(VoidType('void'), Name('main'), [])
25        name:                   Name('main')
26        value or address:       Empty()
27        position:               Pos(Num('6'), Num('5'))
28        size:                   Empty()
29      },
30    Symbol
31      {
32        type qualifier:         Writeable()
33        datatype:               IntType('int')
34        name:                   Name('var@main')
35        value or address:       Num('0')
36        position:               Pos(Num('7'), Num('6'))
37        size:                   Num('1')
38      },
39    Symbol
40      {
41        type qualifier:         Writeable()
42        datatype:               IntType('int')
43        name:                   Name('var@fun2')
44        value or address:       Num('0')
45        position:               Pos(Num('11'), Num('13'))
46        size:                   Num('1')
47      }
48  ]
```

Code 0.58: Symboltabelle für Funktionen, wobei eine Funktion vorher deklariert werden muss

0.0.5.3 Funktionsaufruf

0.0.5.3.1 Ohne Rückgabewert

```
1  struct st {int attr1; int attr2[2];};
2
3  void stack_fun(struct st param[2][3]);
4
5  void main() {
6    struct st local_var[2][3];
7    stack_fun(local_var);
8    return;
9  }
10
11 void stack_fun(struct st param[2][3]) {
12   int local_var;
13 }
```

Code 0.59: PicoC Code für Funktionsaufruf ohne Rückgabewert

```
File
  Name './example_fun_call_no_return_value.picoc_mon',
  [
    Block
      Name 'main.1',
      [
        // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], StructSpec(Name('st'))),
        ↪  Name('local_var')))
        // Exp(Call(Name('stack_fun'), [Name('local_var')]))
        StackMalloc(Num('2'))
        Ref(Global(Num('0')))
        NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
        Exp(GoTo(Name('stack_fun.0')))
        RemoveStackframe()
        Return(Empty())
      ],
    Block
      Name 'stack_fun.0',
      [
        // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('st'))),
        ↪  Name('param')))
        // Exp(Alloc(Writeable(), IntType('int'), Name('local_var')))
        Return(Empty())
      ]
  ]
```

Code 0.60: PicoC Mon Pass für Funktionsaufruf ohne Rückgabewert

```
File
  Name './example_fun_call_no_return_value.reti_blocks',
  [
    Block
      Name 'main.1',
      [
        # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], StructSpec(Name('st'))),
        ↪  Name('local_var')))
        # // Exp(Call(Name('stack_fun'), [Name('local_var')]))
        # StackMalloc(Num('2'))
        SUBI SP 2;
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
        STOREIN SP IN1 1;
        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
        MOVE BAF ACC;
        ADDI SP 3;
        MOVE SP BAF;
        SUBI SP 4;
        STOREIN BAF ACC 0;
        LOADI ACC GoTo(Name('addr@next_instr'));
        ADD ACC CS;
        STOREIN BAF ACC -1;
```

```
25        # Exp(GoTo(Name('stack_fun.0')))
26        Exp(GoTo(Name('stack_fun.0')))
27        # RemoveStackframe()
28        MOVE BAF IN1;
29        LOADIN IN1 BAF 0;
30        MOVE IN1 SP;
31        # Return(Empty())
32        LOADIN BAF PC -1;
33      ],
34    Block
35      Name 'stack_fun.0',
36      [
37        # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('st'))),
          ↪  Name('param')))
38        # // Exp(Alloc(Writeable(), IntType('int'), Name('local_var')))
39        # Return(Empty())
40        LOADIN BAF PC -1;
41      ]
42  ]
```

Code 0.61: RETI Blocks Pass für Funktionsaufruf ohne Rückgabewert

```
1  # // Exp(GoTo(Name('main.1')))
2  # // patched out Exp(GoTo(Name('main.1')))
3  # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], StructSpec(Name('st'))),
   ↪  Name('local_var')))
4  # // Exp(Call(Name('stack_fun'), [Name('local_var')]))
5  # StackMalloc(Num('2'))
6  SUBI SP 2;
7  # Ref(Global(Num('0')))
8  SUBI SP 1;
9  LOADI IN1 0;
10 ADD IN1 DS;
11 STOREIN SP IN1 1;
12 # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
13 MOVE BAF ACC;
14 ADDI SP 3;
15 MOVE SP BAF;
16 SUBI SP 4;
17 STOREIN BAF ACC 0;
18 LOADI ACC 14;
19 ADD ACC CS;
20 STOREIN BAF ACC -1;
21 # Exp(GoTo(Name('stack_fun.0')))
22 JUMP 5;
23 # RemoveStackframe()
24 MOVE BAF IN1;
25 LOADIN IN1 BAF 0;
26 MOVE IN1 SP;
27 # Return(Empty())
28 LOADIN BAF PC -1;
29 # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], StructSpec(Name('st'))), Name('param')))
30 # // Exp(Alloc(Writeable(), IntType('int'), Name('local_var')))
31 # Return(Empty())
```

```
32 LOADIN BAF PC -1;
```

Code 0.62: RETI Pass für Funktionsaufruf ohne Rückgabewert

0.0.5.3.2 Mit Rückgabewert

```
1 void stack_fun() {
2   return 42;
3 }
4
5 void main() {
6   int var = stack_fun();
7 }
```

Code 0.63: PicoC Code für Funktionsaufruf mit Rückgabewert

```
1 File
2   Name './example_fun_call_with_return_value.picoc_mon',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
7         // Return(Num('42'))
8         Exp(Num('42'))
9         Return(Stack(Num('1')))
10      ],
11    Block
12      Name 'main.0',
13      [
14        // Assign(Name('var'), Call(Name('stack_fun'), []))
15        StackMalloc(Num('2'))
16        NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
17        Exp(GoTo(Name('stack_fun.1')))
18        RemoveStackframe()
19        Assign(Global(Num('0')), Stack(Num('1')))
20        Return(Empty())
21      ]
22  ]
```

Code 0.64: PicoC Mon Pass für Funktionsaufruf mit Rückgabewert

```
1 File
2   Name './example_fun_call_with_return_value.reti_blocks',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
```

```
        # // Return(Num('42'))
        # Exp(Num('42'))
        SUBI SP 1;
        LOADI ACC 42;
        STOREIN SP ACC 1;
        # Return(Stack(Num('1')))
        LOADIN SP ACC 1;
        ADDI SP 1;
        LOADIN BAF PC -1;
      ],
    Block
      Name 'main.0',
      [
        # // Assign(Name('var'), Call(Name('stack_fun'), []))
        # StackMalloc(Num('2'))
        SUBI SP 2;
        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
        MOVE BAF ACC;
        ADDI SP 2;
        MOVE SP BAF;
        SUBI SP 2;
        STOREIN BAF ACC 0;
        LOADI ACC GoTo(Name('addr@next_instr'));
        ADD ACC CS;
        STOREIN BAF ACC -1;
        # Exp(GoTo(Name('stack_fun.1')))
        Exp(GoTo(Name('stack_fun.1')))
        # RemoveStackframe()
        MOVE BAF IN1;
        LOADIN IN1 BAF 0;
        MOVE IN1 SP;
        # Assign(Global(Num('0')), Stack(Num('1')))
        LOADIN SP ACC 1;
        STOREIN DS ACC 0;
        ADDI SP 1;
        # Return(Empty())
        LOADIN BAF PC -1;
      ]
  ]
```

Code 0.65: RETI Blocks Pass für Funktionsaufruf mit Rückgabewert

```
# // Exp(GoTo(Name('main.0')))
JUMP 7;
# // Return(Num('42'))
# Exp(Num('42'))
SUBI SP 1;
LOADI ACC 42;
STOREIN SP ACC 1;
# Return(Stack(Num('1')))
LOADIN SP ACC 1;
ADDI SP 1;
LOADIN BAF PC -1;
# // Assign(Name('var'), Call(Name('stack_fun'), []))
```

```
# StackMalloc(Num('2'))
SUBI SP 2;
# NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
MOVE BAF ACC;
ADDI SP 2;
MOVE SP BAF;
SUBI SP 2;
STOREIN BAF ACC 0;
LOADI ACC 17;
ADD ACC CS;
STOREIN BAF ACC -1;
# Exp(GoTo(Name('stack_fun.1')))
JUMP -15;
# RemoveStackframe()
MOVE BAF IN1;
LOADIN IN1 BAF 0;
MOVE IN1 SP;
# Assign(Global(Num('0')), Stack(Num('1')))
LOADIN SP ACC 1;
STOREIN DS ACC 0;
ADDI SP 1;
# Return(Empty())
LOADIN BAF PC -1;
```

Code 0.66: RETI Pass für Funktionsaufruf mit Rückgabewert

0.0.5.3.3 Umsetzung von Call by Sharing für Arrays

```
void stack_fun(int (*param1)[3], int param2[2][3]) {
}

void main() {
  int local_var1[2][3];
  int local_var2[2][3];
  stack_fun(local_var1, local_var2);
}
```

Code 0.67: PicoC Code für Call by Sharing für Arrays

```
File
  Name './example_fun_call_by_sharing_array.picoc_mon',
  [
    Block
      Name 'stack_fun.1',
      [
        // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
        ↪  Name('param1')))
        // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')), Name('param2')))
        Return(Empty())
      ],
    Block
```

```
12      Name 'main.0',
13      [
14        // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
          ↪  Name('local_var1')))
15        // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
          ↪  Name('local_var2')))
16        // Exp(Call(Name('stack_fun'), [Name('local_var1'), Name('local_var2')]))
17        StackMalloc(Num('2'))
18        Ref(Global(Num('0')))
19        Ref(Global(Num('6')))
20        NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
21        Exp(GoTo(Name('stack_fun.1')))
22        RemoveStackframe()
23        Return(Empty())
24      ]
25  ]
```

Code 0.68: PicoC Mon Pass für Call by Sharing für Arrays

```
1 SymbolTable
2   [
3     Symbol
4       {
5         type qualifier:         Empty()
6         datatype:               FunDecl(VoidType('void'), Name('stack_fun'),
          ↪  [Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int'))),
          ↪  Name('param1')), Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')),
          ↪  Name('param2'))])
7         name:                   Name('stack_fun')
8         value or address:       Empty()
9         position:               Pos(Num('1'), Num('5'))
10        size:                   Empty()
11      },
12    Symbol
13      {
14        type qualifier:         Writeable()
15        datatype:               PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
16        name:                   Name('param1@stack_fun')
17        value or address:       Num('0')
18        position:               Pos(Num('1'), Num('21'))
19        size:                   Num('1')
20      },
21    Symbol
22      {
23        type qualifier:         Writeable()
24        datatype:               PntrDecl(Num('1'), ArrayDecl([Num('3')], IntType('int')))
25        name:                   Name('param2@stack_fun')
26        value or address:       Num('1')
27        position:               Pos(Num('1'), Num('37'))
28        size:                   Num('1')
29      },
30    Symbol
31      {
32        type qualifier:         Empty()
```

```
      datatype:               FunDecl(VoidType('void'), Name('main'), [])
      name:                   Name('main')
      value or address:       Empty()
      position:               Pos(Num('4'), Num('5'))
      size:                   Empty()
    },
  Symbol
    {
      type qualifier:         Writeable()
      datatype:               ArrayDecl([Num('2'), Num('3')], IntType('int'))
      name:                   Name('local_var1@main')
      value or address:       Num('0')
      position:               Pos(Num('5'), Num('6'))
      size:                   Num('6')
    },
  Symbol
    {
      type qualifier:         Writeable()
      datatype:               ArrayDecl([Num('2'), Num('3')], IntType('int'))
      name:                   Name('local_var2@main')
      value or address:       Num('6')
      position:               Pos(Num('6'), Num('6'))
      size:                   Num('6')
    }
]
```

Code 0.69: Symboltabelle für Call by Sharing für Arrays

```
File
  Name './example_fun_call_by_sharing_array.reti_blocks',
  [
    Block
      Name 'stack_fun.1',
      [
        # // Exp(Alloc(Writeable(), PntrDecl(Num('1'), ArrayDecl([Num('3')],
        ↪  IntType('int'))), Name('param1')))
        # // Exp(Alloc(Writeable(), ArrayDecl([Num('3')], IntType('int')), Name('param2')))
        # Return(Empty())
        LOADIN BAF PC -1;
      ],
    Block
      Name 'main.0',
      [
        # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
        ↪  Name('local_var1')))
        # // Exp(Alloc(Writeable(), ArrayDecl([Num('2'), Num('3')], IntType('int')),
        ↪  Name('local_var2')))
        # // Exp(Call(Name('stack_fun'), [Name('local_var1'), Name('local_var2')]))
        # StackMalloc(Num('2'))
        SUBI SP 2;
        # Ref(Global(Num('0')))
        SUBI SP 1;
        LOADI IN1 0;
        ADD IN1 DS;
```

```
24        STOREIN SP IN1 1;
25        # Ref(Global(Num('6')))
26        SUBI SP 1;
27        LOADI IN1 6;
28        ADD IN1 DS;
29        STOREIN SP IN1 1;
30        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
31        MOVE BAF ACC;
32        ADDI SP 4;
33        MOVE SP BAF;
34        SUBI SP 4;
35        STOREIN BAF ACC 0;
36        LOADI ACC GoTo(Name('addr@next_instr'));
37        ADD ACC CS;
38        STOREIN BAF ACC -1;
39        # Exp(GoTo(Name('stack_fun.1')))
40        Exp(GoTo(Name('stack_fun.1')))
41        # RemoveStackframe()
42        MOVE BAF IN1;
43        LOADIN IN1 BAF 0;
44        MOVE IN1 SP;
45        # Return(Empty())
46        LOADIN BAF PC -1;
47      ]
48  ]
```

Code 0.70: RETI Block Pass für Call by Sharing für Arrays

0.0.5.3.4 Umsetzung von Call by Value für Structs

```
1 struct st {int attr1; int attr2[2];};
2
3 void stack_fun(struct st param) {
4 }
5
6 void main() {
7   struct st local_var;
8   stack_fun(local_var);
9 }
```

Code 0.71: PicoC Code für Call by Value für Structs

```
1 File
2   Name './example_fun_call_by_value_struct.picoc_mon',
3   [
4     Block
5       Name 'stack_fun.1',
6       [
7         // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('param')))
8         Return(Empty())
9       ],
```

```
    Block
      Name 'main.0',
      [
        // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('local_var')))
        // Exp(Call(Name('stack_fun'), [Name('local_var')]))
        StackMalloc(Num('2'))
        Assign(Stack(Num('3')), Global(Num('0')))
        NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
        Exp(GoTo(Name('stack_fun.1')))
        RemoveStackframe()
        Return(Empty())
      ]
  ]
```

Code 0.72: PicoC Mon Pass für Call by Value für Structs

```
File
  Name './example_fun_call_by_value_struct.reti_blocks',
  [
    Block
      Name 'stack_fun.1',
      [
        # // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('param')))
        # Return(Empty())
        LOADIN BAF PC -1;
      ],
    Block
      Name 'main.0',
      [
        # // Exp(Alloc(Writeable(), StructSpec(Name('st')), Name('local_var')))
        # // Exp(Call(Name('stack_fun'), [Name('local_var')]))
        # StackMalloc(Num('2'))
        SUBI SP 2;
        # Assign(Stack(Num('3')), Global(Num('0')))
        SUBI SP 3;
        LOADIN DS ACC 0;
        STOREIN SP ACC 1;
        LOADIN DS ACC 1;
        STOREIN SP ACC 2;
        LOADIN DS ACC 2;
        STOREIN SP ACC 3;
        # NewStackframe(Name('stack_fun'), GoTo(Name('addr@next_instr')))
        MOVE BAF ACC;
        ADDI SP 5;
        MOVE SP BAF;
        SUBI SP 5;
        STOREIN BAF ACC 0;
        LOADI ACC GoTo(Name('addr@next_instr'));
        ADD ACC CS;
        STOREIN BAF ACC -1;
        # Exp(GoTo(Name('stack_fun.1')))
        Exp(GoTo(Name('stack_fun.1')))
        # RemoveStackframe()
        MOVE BAF IN1;
```

```
39          LOADIN IN1 BAF 0;
40          MOVE IN1 SP;
41          # Return(Empty())
42          LOADIN BAF PC -1;
43        ]
44    ]
```

Code 0.73: RETI Block Pass für Call by Value für Structs

0.0.6 Umsetzung kleinerer Details

0.1 Fehlermeldungen

0.1.1 Error Handler

0.1.2 Arten von Fehlermeldungen

0.1.2.1 Syntaxfehler

0.1.2.2 Laufzeitfehler

Literatur

Online

- *GCC, the GNU Compiler Collection - GNU Project.* URL: https://gcc.gnu.org/ (besucht am 13. 07. 2022).