Informatik II für Verkehrsingenieure
C_0 (Kapitel 3)

Janis Voigtländer

Technische Universität Dresden

Sommersemester 2007

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)
- keine Konstantendeklarationen

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)
- keine Konstantendeklarationen
- keine Kommentare

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)
- keine Konstantendeklarationen
- keine Kommentare
- lediglich `while`-Schleifen (ohne `break`) und `if`-Verzweigungen als Kontrollstrukturen

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)
- keine Konstantendeklarationen
- keine Kommentare
- lediglich `while`-Schleifen (ohne `break`) und `if`-Verzweigungen als Kontrollstrukturen

Vorhanden:

- Ein- und Ausgabe

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)
- keine Konstantendeklarationen
- keine Kommentare
- lediglich `while`-Schleifen (ohne `break`) und `if`-Verzweigungen als Kontrollstrukturen

Vorhanden:

- Ein- und Ausgabe
- Zuweisungen

Von C zu C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)
- keine Konstantendeklarationen
- keine Kommentare
- lediglich `while`-Schleifen (ohne `break`) und `if`-Verzweigungen als Kontrollstrukturen

Vorhanden:

- Ein- und Ausgabe
- Zuweisungen
- arithmetische und Vergleichsoperationen

Beispiel

```
#include<stdio.h>
int main()
{ int i,n,s;
  scanf("%i",&n);
  i=1;
  s=0;
  while (i<=n)
   { s=s+i*i;
     i=i+1;
   }
  printf("%d",s);
  return 0;
}
```

Beispiel

```
#include<stdio.h>
int main()
{ int i,n,s;
  scanf("%i",&n);
  i=1;
  s=0;
  while (i<=n)
   { s=s+i*i;
     i=i+1;
   }
  printf("%d",s);
  return 0;
}
```

Formale Beschreibung: EBNF

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen,

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen,

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation,

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{} \cdots \hat{\}}$ (Wiederholungsklammern),

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation,
  $\hat{\{}\cdots\hat{\}}$ (Wiederholungsklammern),
  $\hat{[}\cdots\hat{]}$ (Optionsklammern),

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{}\cdots\hat{\}}$ (Wiederholungsklammern), $\hat{[}\cdots\hat{]}$ (Optionsklammern), $\hat{|}$ (Alternativstrich),

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{}\cdots\hat{\}}$ (Wiederholungsklammern), $\hat{[}\cdots\hat{]}$ (Optionsklammern), $\hat{|}$ (Alternativstrich), $\hat{(}\cdots\hat{)}$ (Vorrangsklammern)

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{}\cdots\hat{\}}$ (Wiederholungsklammern), $\hat{[}\cdots\hat{]}$ (Optionsklammern), $\hat{|}$ (Alternativstrich), $\hat{(}\cdots\hat{)}$ (Vorrangsklammern)
- Festlegung einer syntaktischen Variable als Startsymbol

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{} \cdots \hat{\}}$ (Wiederholungsklammern), $\hat{[} \cdots \hat{]}$ (Optionsklammern), $\hat{|}$ (Alternativstrich), $\hat{(} \cdots \hat{)}$ (Vorrangsklammern)
- Festlegung einer syntaktischen Variable als Startsymbol

Beispiel: $\mathcal{E} = (V, \Sigma, S, R)$ mit:

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{} \cdots \hat{\}}$ (Wiederholungsklammern), $\hat{[} \cdots \hat{]}$ (Optionsklammern), $\hat{|}$ (Alternativstrich), $\hat{(} \cdots \hat{)}$ (Vorrangsklammern)
- Festlegung einer syntaktischen Variable als Startsymbol

Beispiel: $\mathcal{E} = (V, \Sigma, S, R)$ mit:

- $V = \{S, A, B\}$

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{} \cdots \hat{\}}$ (Wiederholungsklammern), $\hat{[} \cdots \hat{]}$ (Optionsklammern), $\hat{|}$ (Alternativstrich), $\hat{(} \cdots \hat{)}$ (Vorrangsklammern)
- Festlegung einer syntaktischen Variable als Startsymbol

Beispiel: $\mathcal{E} = (V, \Sigma, S, R)$ mit:

- $V = \{S, A, B\}$
- $\Sigma = \{a, b\}$

Wiederholung — EBNF

Ideen:

- Definition syntaktischer Variablen über Regeln
- für jede syntaktische Variable genau eine Regel
- rechte Regelseiten aufgebaut aus syntaktischen Variablen, Terminalsymbolen, Konkatenation, $\hat{\{}\cdots\hat{\}}$ (Wiederholungsklammern), $\hat{[}\cdots\hat{]}$ (Optionsklammern), $\hat{|}$ (Alternativstrich), $\hat{(}\cdots\hat{)}$ (Vorrangsklammern)
- Festlegung einer syntaktischen Variable als Startsymbol

Beispiel: $\mathcal{E} = (V, \Sigma, S, R)$ mit:

- $V = \{S, A, B\}$
- $\Sigma = \{a, b\}$
- R : $S ::= A\hat{[}B\hat{]}\,\hat{|}\,BAA.$
  $A ::= \hat{\{}b\hat{\}}a.$
  $B ::= b\,\hat{|}\,bB.$

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= { $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;`}.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= { $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;` }.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ $\hat{\{}$,⟨Ident⟩$\hat{\}}$;.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= `{` $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;}`.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ $\hat{\{}$`,`⟨Ident⟩$\hat{\}}$`;`.

⟨StatementSequence⟩ ::= ⟨Statement⟩ $\hat{\{}$⟨Statement⟩$\hat{\}}$.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= `{` $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;}`.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ $\hat{\{}$`,`⟨Ident⟩$\hat{\}}$`;`.

⟨StatementSequence⟩ ::= ⟨Statement⟩ $\hat{\{}$⟨Statement⟩$\hat{\}}$.

⟨Statement⟩ ::= ⟨Assignment⟩ $\hat{|}$ ⟨IfStatement⟩ $\hat{|}$ ⟨WhileStatement⟩ $\hat{|}$
`scanf("%i", &`⟨Ident⟩`);` $\hat{|}$ `printf("%d",`⟨Ident⟩`);` $\hat{|}$
⟨CompStatement⟩.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= `{` $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;}`.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ $\hat{\{}$`,`⟨Ident⟩$\hat{\}}$`;`.

⟨StatementSequence⟩ ::= ⟨Statement⟩ $\hat{\{}$⟨Statement⟩$\hat{\}}$.

⟨Statement⟩ ::= ⟨Assignment⟩ $\hat{|}$ ⟨IfStatement⟩ $\hat{|}$ ⟨WhileStatement⟩ $\hat{|}$
`scanf("%i", &`⟨Ident⟩`);` $\hat{|}$ `printf("%d",`⟨Ident⟩`);` $\hat{|}$
⟨CompStatement⟩.

⟨Assignment⟩ ::= ⟨Ident⟩ = ⟨SimpleExpression⟩`;`.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= `{` $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;}`.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ $\hat{\{}$`,`⟨Ident⟩$\hat{\}}$`;`.

⟨StatementSequence⟩ ::= ⟨Statement⟩ $\hat{\{}$⟨Statement⟩$\hat{\}}$.

⟨Statement⟩ ::= ⟨Assignment⟩ $\hat{|}$ ⟨IfStatement⟩ $\hat{|}$ ⟨WhileStatement⟩ $\hat{|}$
`scanf("%i", &`⟨Ident⟩`);` $\hat{|}$ `printf("%d",`⟨Ident⟩`);` $\hat{|}$
⟨CompStatement⟩.

⟨Assignment⟩ ::= ⟨Ident⟩ = ⟨SimpleExpression⟩`;`.

⟨IfStatement⟩ ::= `if (`⟨BoolExpression⟩`)` ⟨Statement⟩
$\hat{[}$`else` ⟨Statement⟩$\hat{]}$.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= `{` $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;}`.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ $\hat{\{}$`,`⟨Ident⟩$\hat{\}}$`;`.

⟨StatementSequence⟩ ::= ⟨Statement⟩ $\hat{\{}$⟨Statement⟩$\hat{\}}$.

⟨Statement⟩ ::= ⟨Assignment⟩ $\hat{|}$ ⟨IfStatement⟩ $\hat{|}$ ⟨WhileStatement⟩ $\hat{|}$
`scanf("%i", &`⟨Ident⟩`);` $\hat{|}$ `printf("%d",`⟨Ident⟩`);` $\hat{|}$
⟨CompStatement⟩.

⟨Assignment⟩ ::= ⟨Ident⟩ `=` ⟨SimpleExpression⟩`;`.

⟨IfStatement⟩ ::= `if (`⟨BoolExpression⟩`)` ⟨Statement⟩
$\hat{[}$`else` ⟨Statement⟩$\hat{]}$.

⟨WhileStatement⟩ ::= `while (`⟨BoolExpression⟩`)` ⟨Statement⟩.

EBNF-Definition für C_0: Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= `{` $\hat{[}$⟨VarDeclaration⟩$\hat{]}$ $\hat{[}$⟨StatementSequence⟩$\hat{]}$ `return 0;}`.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ $\hat{\{}$`,`⟨Ident⟩$\hat{\}}$`;`.

⟨StatementSequence⟩ ::= ⟨Statement⟩ $\hat{\{}$⟨Statement⟩$\hat{\}}$.

⟨Statement⟩ ::= ⟨Assignment⟩ $\hat{|}$ ⟨IfStatement⟩ $\hat{|}$ ⟨WhileStatement⟩ $\hat{|}$
`scanf("%i", &`⟨Ident⟩`);` $\hat{|}$ `printf("%d",`⟨Ident⟩`);` $\hat{|}$
⟨CompStatement⟩.

⟨Assignment⟩ ::= ⟨Ident⟩ `=` ⟨SimpleExpression⟩`;`.

⟨IfStatement⟩ ::= `if (`⟨BoolExpression⟩`)` ⟨Statement⟩
$\hat{[}$`else` ⟨Statement⟩$\hat{]}$.

⟨WhileStatement⟩ ::= `while (`⟨BoolExpression⟩`)` ⟨Statement⟩.

⟨CompStatement⟩ ::= `{`⟨StatementSequence⟩`}`.

EBNF-Definition für C_0: Ausdrücke

⟨BoolExpression⟩ ::= ⟨SimpleExpression⟩ ⟨Relation⟩ ⟨SimpleExpression⟩.

EBNF-Definition für C_0: Ausdrücke

⟨BoolExpression⟩ ::= ⟨SimpleExpression⟩ ⟨Relation⟩ ⟨SimpleExpression⟩.

⟨Relation⟩ ::= == | != | < | > | <= | >=.

EBNF-Definition für C_0: Ausdrücke

⟨BoolExpression⟩ ::= ⟨SimpleExpression⟩ ⟨Relation⟩ ⟨SimpleExpression⟩.

⟨Relation⟩ ::= == | != | < | > | <= | >=.

⟨SimpleExpression⟩ ::= [+ | -] ⟨Term⟩ { (+ | -) ⟨Term⟩ }.

EBNF-Definition für C_0: Ausdrücke

⟨BoolExpression⟩ ::= ⟨SimpleExpression⟩ ⟨Relation⟩ ⟨SimpleExpression⟩.

⟨Relation⟩ ::= == | != | < | > | <= | >=.

⟨SimpleExpression⟩ ::= [+ | -] ⟨Term⟩ { (+ | -) ⟨Term⟩ }.

⟨Term⟩ ::= ⟨Factor⟩ { (* | / | %) ⟨Factor⟩ }.

EBNF-Definition für C_0: Ausdrücke

⟨BoolExpression⟩ ::= ⟨SimpleExpression⟩ ⟨Relation⟩ ⟨SimpleExpression⟩.

⟨Relation⟩ ::= == | != | < | > | <= | >=.

⟨SimpleExpression⟩ ::= [+ | -] ⟨Term⟩ { (+ | -) ⟨Term⟩ }.

⟨Term⟩ ::= ⟨Factor⟩ { (* | / | %) ⟨Factor⟩ }.

⟨Factor⟩ ::= ⟨Ident⟩ | ⟨Number⟩ | (⟨SimpleExpression⟩).

EBNF-Definition für C_0: Ausdrücke

⟨BoolExpression⟩ ::= ⟨SimpleExpression⟩ ⟨Relation⟩ ⟨SimpleExpression⟩.

⟨Relation⟩ ::= == | != | < | > | <= | >=.

⟨SimpleExpression⟩ ::= [+ | -] ⟨Term⟩ { (+ | -) ⟨Term⟩ }.

⟨Term⟩ ::= ⟨Factor⟩ { (* | / | %) ⟨Factor⟩ }.

⟨Factor⟩ ::= ⟨Ident⟩ | ⟨Number⟩ | (⟨SimpleExpression⟩).

⟨Ident⟩ ::= ⟨Letter⟩ { ⟨Letter⟩ | ⟨Digit⟩ }.

⟨Letter⟩ ::= A | B | C | ··· | Z | a | b | ··· | z.

⟨Digit⟩ ::= 0 | 1 | ··· | 9.

⟨Number⟩ ::= 0 | (⟨Pdigit⟩ {⟨Digit⟩}).

⟨Pdigit⟩ ::= 1 | 2 | ··· | 9.

Wiederholung — AM_0

$AM_0 = BZ \times DK \times HS \times \underline{Inp} \times \underline{Out}$ mit:

BZ	$= \mathbb{N}$	Befehlszähler
DK	$= \mathbb{Z}^*$	Datenkeller
HS	$= \{h \mid h : \mathbb{N} \rightharpoonup \mathbb{Z}\}$	Hauptspeicher
$\underline{Inp}$	$= \mathbb{Z}^*$	Eingabeband
$\underline{Out}$	$= \mathbb{Z}^*$	Ausgabeband

- READ n: Lesen von Eingabeband in Hauptspeicher
- WRITE n: Ausgabe aus Hauptspeicher auf Ausgabeband
- LOAD n: Ablage aus Hauptspeicher auf Datenkeller
- STORE n: Entnahme aus Datenkeller in Hauptspeicher
- LIT z: Ablage einer Konstante auf Datenkeller
- ADD, MUL, SUB, DIV, MOD, LT, EQ, NE, GT, LE, GE: Berechnungen und Vergleiche (auf Datenkeller)
- JMP n: Sprung
- JMC n: Sprung abhängig von Datenkeller

Wiederholung — C_0

Motivation:

- Wir wollen C-Programme in AM_0-Programme übersetzen, ...
- um die Übersetzung einfach zu halten, jedoch nicht alle Sprachkonstrukte betrachten.

Einschränkungen:

- nur `int` als Typ
- keine Funktionsdefinitionen (außer `main`)
- keine Konstantendeklarationen
- keine Kommentare
- lediglich `while`-Schleifen (ohne `break`) und `if`-Verzweigungen als Kontrollstrukturen

Vorhanden:

- Ein- und Ausgabe
- Zuweisungen
- arithmetische und Vergleichsoperationen

Wiederholung — EBNF-Definition der Programmstruktur

⟨Program⟩ ::= `#include <stdio.h>`
`int main()`
⟨Block⟩.

⟨Block⟩ ::= `{` [⟨VarDeclaration⟩] [⟨StatementSequence⟩] `return 0;}`.

⟨VarDeclaration⟩ ::= `int` ⟨Ident⟩ {`,`⟨Ident⟩}`;`.

⟨StatementSequence⟩ ::= ⟨Statement⟩ {⟨Statement⟩}.

⟨Statement⟩ ::= ⟨Assignment⟩ | ⟨IfStatement⟩ | ⟨WhileStatement⟩ |
`scanf("%i", &`⟨Ident⟩`);` | `printf("%d",`⟨Ident⟩`);` |
⟨CompStatement⟩.

⟨Assignment⟩ ::= ⟨Ident⟩ = ⟨SimpleExpression⟩`;`.

⟨IfStatement⟩ ::= `if (`⟨BoolExpression⟩`)` ⟨Statement⟩
[`else` ⟨Statement⟩].

⟨WhileStatement⟩ ::= `while (`⟨BoolExpression⟩`)` ⟨Statement⟩.

⟨CompStatement⟩ ::= `{`⟨StatementSequence⟩`}`.

Wiederholung — EBNF-Definition für Ausdrücke

⟨BoolExpression⟩ ::= ⟨SimpleExpression⟩ ⟨Relation⟩ ⟨SimpleExpression⟩.

⟨Relation⟩ ::= == | != | < | > | <= | >=.

⟨SimpleExpression⟩ ::= [+ | -] ⟨Term⟩ { (+ | -) ⟨Term⟩ }.

⟨Term⟩ ::= ⟨Factor⟩ { (* | / | %) ⟨Factor⟩ }.

⟨Factor⟩ ::= ⟨Ident⟩ | ⟨Number⟩ | (⟨SimpleExpression⟩).

⟨Ident⟩ ::= ⟨Letter⟩ { ⟨Letter⟩ | ⟨Digit⟩ }.

⟨Letter⟩ ::= A | B | C | ··· | Z | a | b | ··· | z.

⟨Digit⟩ ::= 0 | 1 | ··· | 9.

⟨Number⟩ ::= 0 | (⟨Pdigit⟩ {⟨Digit⟩}).

⟨Pdigit⟩ ::= 1 | 2 | ··· | 9.

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

- Jeder Bezeichner darf höchstens einmal in $\langle \text{VarDeclaration} \rangle$ deklariert sein.

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

- Jeder Bezeichner darf höchstens einmal in $\langle$VarDeclaration$\rangle$ deklariert sein.
- Wenn ein Bezeichner in der $\langle$StatementSequence$\rangle$ des $\langle$Block$\rangle$s auftritt, so muß er in der $\langle$VarDeclaration$\rangle$ des $\langle$Block$\rangle$s deklariert sein.

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

- Jeder Bezeichner darf höchstens einmal in $\langle \text{VarDeclaration} \rangle$ deklariert sein.
- Wenn ein Bezeichner in der $\langle \text{StatementSequence} \rangle$ des $\langle \text{Block} \rangle$s auftritt, so muß er in der $\langle \text{VarDeclaration} \rangle$ des $\langle \text{Block} \rangle$s deklariert sein.

Übersetzungsstrategie:

- syntaxgesteuert

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

- Jeder Bezeichner darf höchstens einmal in $\langle \text{VarDeclaration} \rangle$ deklariert sein.
- Wenn ein Bezeichner in der $\langle \text{StatementSequence} \rangle$ des $\langle \text{Block} \rangle$s auftritt, so muß er in der $\langle \text{VarDeclaration} \rangle$ des $\langle \text{Block} \rangle$s deklariert sein.

Übersetzungsstrategie:

- syntaxgesteuert, das heißt:
- Übersetzung von $W(\langle \text{Program} \rangle)$ benutzt Übersetzung von $W(\langle \text{Block} \rangle)$.

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

- Jeder Bezeichner darf höchstens einmal in $\langle \text{VarDeclaration} \rangle$ deklariert sein.
- Wenn ein Bezeichner in der $\langle \text{StatementSequence} \rangle$ des $\langle \text{Block} \rangle$s auftritt, so muß er in der $\langle \text{VarDeclaration} \rangle$ des $\langle \text{Block} \rangle$s deklariert sein.

Übersetzungsstrategie:

- syntaxgesteuert, das heißt:
- Übersetzung von $W(\langle \text{Program} \rangle)$ benutzt Übersetzung von $W(\langle \text{Block} \rangle)$.
- Übersetzung von $W(\langle \text{Block} \rangle)$ benutzt Analyse von $W(\langle \text{VarDeclaration} \rangle)$ und Übersetzung von $W(\langle \text{StatementSequence} \rangle)$.

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

- Jeder Bezeichner darf höchstens einmal in $\langle \text{VarDeclaration} \rangle$ deklariert sein.
- Wenn ein Bezeichner in der $\langle \text{StatementSequence} \rangle$ des $\langle \text{Block} \rangle$s auftritt, so muß er in der $\langle \text{VarDeclaration} \rangle$ des $\langle \text{Block} \rangle$s deklariert sein.

Übersetzungsstrategie:

- syntaxgesteuert, das heißt:
- Übersetzung von $W(\langle \text{Program} \rangle)$ benutzt Übersetzung von $W(\langle \text{Block} \rangle)$.
- Übersetzung von $W(\langle \text{Block} \rangle)$ benutzt Analyse von $W(\langle \text{VarDeclaration} \rangle)$ und Übersetzung von $W(\langle \text{StatementSequence} \rangle)$.
- Übersetzung von $W(\langle \text{StatementSequence} \rangle)$ benutzt Übersetzung von $W(\langle \text{Statement} \rangle)$.

Eingabesprache für die Übersetzung

$W(\mathcal{E}, \langle \text{Program} \rangle)$, mit zwei Nebenbedingungen:

- Jeder Bezeichner darf höchstens einmal in $\langle \text{VarDeclaration} \rangle$ deklariert sein.
- Wenn ein Bezeichner in der $\langle \text{StatementSequence} \rangle$ des $\langle \text{Block} \rangle$s auftritt, so muß er in der $\langle \text{VarDeclaration} \rangle$ des $\langle \text{Block} \rangle$s deklariert sein.

Übersetzungsstrategie:

- syntaxgesteuert, das heißt:
- Übersetzung von $W(\langle \text{Program} \rangle)$ benutzt Übersetzung von $W(\langle \text{Block} \rangle)$.
- Übersetzung von $W(\langle \text{Block} \rangle)$ benutzt Analyse von $W(\langle \text{VarDeclaration} \rangle)$ und Übersetzung von $W(\langle \text{StatementSequence} \rangle)$.
- Übersetzung von $W(\langle \text{StatementSequence} \rangle)$ benutzt Übersetzung von $W(\langle \text{Statement} \rangle)$.
- ...