

Matrikelnummer:			
 DHBW Duale Hochschule Baden-Württemberg Stuttgart ÜBUNGSKLAUSUR	Fakultät	Technik	
	Studiengang:	Angewandte Informatik	
	Jahrgang / Kurs :	2014 / 14B	
	Studienhalbjahr:	4. Semester	
Datum:	18. Mai 2016	Bearbeitungszeit:	60 Minuten
Modul:	TINF2002.3	Dozent:	Stephan Schulz
Unit:	Compilerbau		
Hilfsmittel:	Vorlesungsskript, eigene Notizen		
Punkte:		Note:	

Aufgabe	erreichbar	erreicht
1	5	
2	10	
3	12	
4	10	
5	7	
6	9	
Summe	53	

1. Sind Sie gesund und prüfungsfähig?
2. Sind Ihre Taschen und sämtliche Unterlagen, insbesondere alle nicht erlaubten Hilfsmittel, seitlich an der Wand zum Gang hin abgestellt und nicht in Reichweite des Arbeitsplatzes?
3. Haben Sie auch außerhalb des Klausorraumes im Gebäude keine unerlaubten Hilfsmittel oder ähnliche Unterlagen liegen lassen?
4. Haben Sie Ihr Handy ausgeschaltet und abgegeben?

(Falls Ziff. 2 oder 3 nicht erfüllt sind, liegt ein Täuschungsversuch vor, der die Note „nicht ausreichend“ zur Folge hat.)

Aufgabe 1 (5 Punkte)

Beschreiben Sie jeweils kurz die Funktion eines Parsers in einem Compiler. Wo kommt er zum Einsatz, und was sind seine Ein- und Ausgaben.

Lösung: Ein Parser überprüft und erkennt die syntaktische Struktur eines Programmes. Er wandelt eine Folge von Token (Terminalsymbole der kontextfreien Grammatik) in eine Zwischendarstellung (z.B. einen Parse Tree oder direkt einen abstrakten Syntaxbaum) um. Der Parser ist Teil des Front End eines Compilers und liegt zwischen Scanner und semantischer Überprüfung.

Aufgabe 2 (10 Punkte)

Das folgende *nanoLang*-Programm enthält 5 Typen von Fehlern. Markieren Sie mindestens einen Fehler jeden Typs und beschreiben Sie, welche Phase des Compilers den Fehler jeweils identifiziert.

```
Integer square(Int n)
{
    return n*n;
}

String root(Integer n)
{
    Integer guess;
    guess := n;

    while(square(guess)!=n)
    {
        if(square(guess)>n)
        {
            guess := guess/2;
        }
        else if(square(guess)<n)
        {
            guess := guess+1;
        }
    }
    return guess;
}

Integer main(String arg)
{
    print root(arg);
    print "\n";

    return 0;
}
```

Lösung:

- `Int` ist kein korrekter Typ. Parser.
- `:=` ist kein legales Token, `:` kann nicht vorkommen. Lexer.
- Nach `else` muss ein Block kommen, kein einfaches Statement. Parser.
- `return` returned `Integer`, aber Funktion hat Typ `String`. Semantik-Checker.
- `root()` erwartet Argument vom Typ `Integer`, bekommt Typ `String`. Semantik-Checker.

Aufgabe 3 (4+3+5 Punkte)

Betrachten Sie die folgende formale Sprache von *Typen*: `int` und `char` sind Typen. Wenn `t1` und `t2` Typen sind, so auch `<t1, t2>`, `t1->t2`, und `(t1)`. Der Pfeil `->` ist linksassoziativ. Beispiele für korrekte *Typen* und Gegenbeispiele sind:

Korrekte Typen	Keine korrekten Typen
<code>char -> int</code>	<code>int()</code>
<code><int, <int, char>></code>	<code><char></code>
<code>(int -> (int))</code>	<code>real->hans</code>
<code>(int -> (int -> int))</code>	<code>char,int</code>
<code><int -> char, char></code>	<code><int -> char<</code>

a) Geben Sie eine Kontext-freie Grammatik $G = \langle V_N, V_T, P, S \rangle$ an, mit der die Sprache der Typen beschrieben wird. Sie können `int`, `char`, `->`, `<` und `>` als Terminalsymbole voraussetzen. Ihre Grammatik sollte die Linksassoziativität von `->` korrekt reflektieren (d.h. nur Parse-Trees erlauben, die diese Eigenschaft haben).

b) Geben Sie für folgende Worte eine Linksableitung in Ihrer Grammatik ab.

1. `<int, <int, int>>`
2. `(<char, int> -> char)`

c) Geben Sie zu folgenden Worten einen Parse-Tree mit Ihrer Grammatik an.

1. `char -> int`
2. `<int,char->int> -> char -> char`

a) $V_N = \{T, A\}$, $V_T = \{->, <, >, (,), \text{int}, \text{char}\}$, Startsymbol T , P siehe unten.

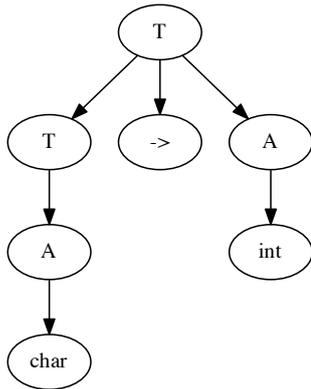
1. $T \rightarrow T -> A \mid A$
2. $A \rightarrow \langle T, T \rangle \mid (T) \mid \text{int} \mid \text{char}$

b)

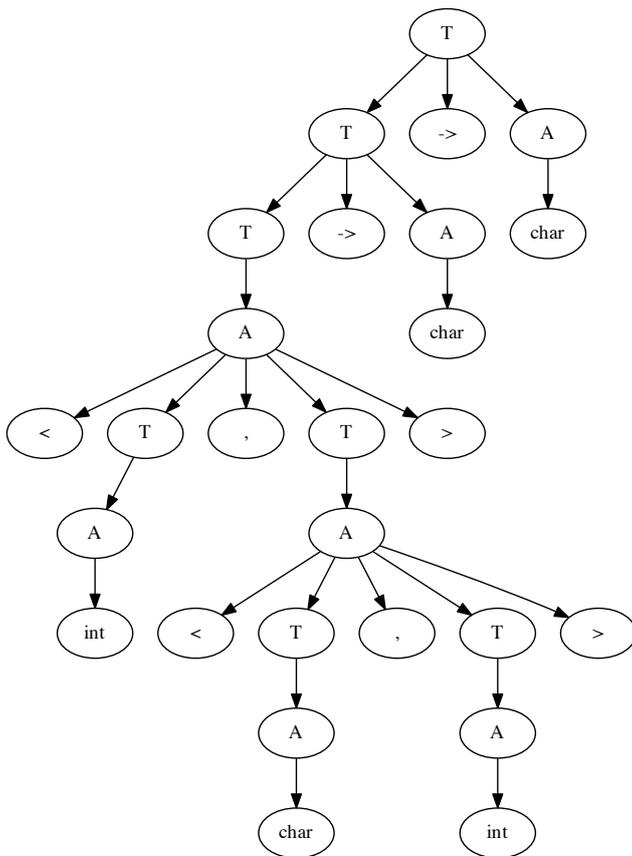
1.
 - $T \Rightarrow A$
 - $\Rightarrow \langle T, T \rangle$
 - $\Rightarrow \langle A, T \rangle$
 - $\Rightarrow \langle \text{int}, T \rangle$
 - $\Rightarrow \langle \text{int}, A \rangle$
 - $\Rightarrow \langle \text{int}, \langle T, T \rangle \rangle$
 - $\Rightarrow \langle \text{int}, \langle A, T \rangle \rangle$
 - $\Rightarrow \langle \text{int}, \langle \text{int}, T \rangle \rangle$
 - $\Rightarrow \langle \text{int}, \langle \text{int}, A \rangle \rangle$
 - $\Rightarrow \langle \text{int}, \langle \text{int}, \text{int} \rangle \rangle$

2.
 - $T \Rightarrow A$
 - $\Rightarrow (T)$
 - $\Rightarrow (T->A)$
 - $\Rightarrow (A->A)$
 - $\Rightarrow (\langle T, T \rangle ->A)$
 - $\Rightarrow (\langle A, T \rangle ->A)$
 - $\Rightarrow (\langle \text{char}, T \rangle ->A)$
 - $\Rightarrow (\langle \text{char}, A \rangle ->A)$
 - $\Rightarrow (\langle \text{char}, \text{int} \rangle ->A)$
 - $\Rightarrow (\langle \text{char}, \text{int} \rangle ->\text{char})$

Fortsetzung



c) 1.

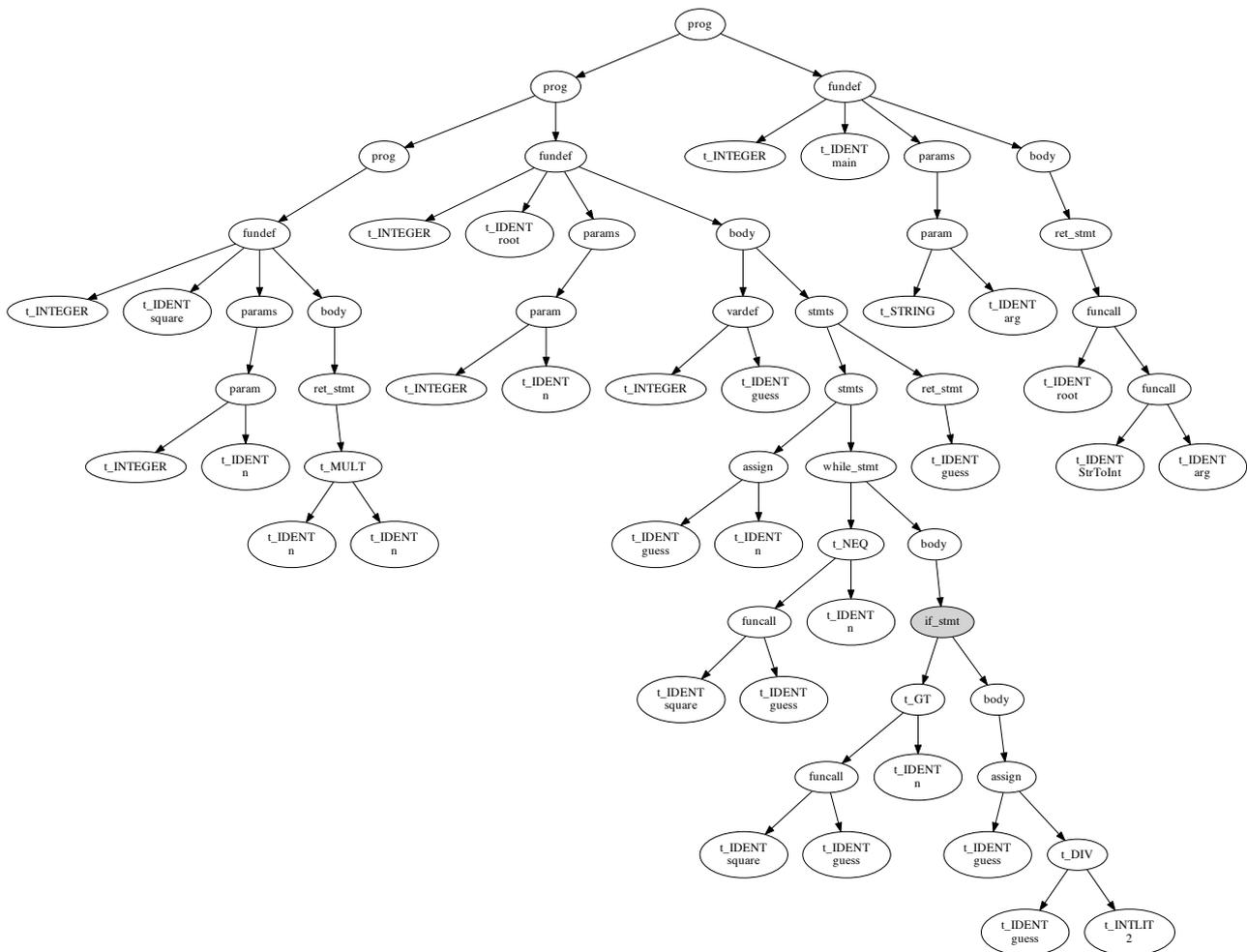


2.

Aufgabe 4 (6+4 Punkte)

Betrachten Sie den *abstract syntax tree* (AST) auf der nächsten Seite.

- a) Generieren Sie aus dem AST ein äquivalentes *nanoLang*-Programm. Hinweis: Das Programm sollte bei normaler Schriftgröße problemlos unter den AST passen.
- b) Nehmen Sie an, dass das Programm vollständig geparkt ist und alle Symbole in die Symboltabellen eingetragen sind. Welche Symboltabellen sind an dem grau hinterlegten Knoten `it_stmt` gültig? Geben Sie die Hierarchie und die Einträge mit Namen und Typ an. Geben Sie nur die Namen an, die auch im Programm vorkommen, keine nicht verwendeten Library-Funktionen.



Lösung:

a) Programm

```
Integer square(Integer n)
{
    return n*n;
}
```

```
Integer root(Integer n)
{
    Integer guess;
    guess = n;

    while(square(guess)!=n)
    {
        if(square(guess)>n)
        {
            guess = guess/2;
        }
    }
    return guess;
}
```

```
Integer main(String arg)
{
    return root(StrToInt(arg));
}
```

b) Symboltabellen:

- Global

- main():String->Integer
- root():Integer->Integer
- square():Integer->Integer
- StrToInt():String->Integer

- root() Header

- n: Integer

- root() Body

- guess: Integer

- Body while

- Leer

Aufgabe 5 (7 Punkte)

Betrachten Sie die folgende Grammatik: $G = \langle V_N, V_T, P, E \rangle$ mit $V_N = \{E\}$, $V_T = \{v, n, *, +, (,)\}$, und folgenden Produktionen in P :

- $E \rightarrow (E)$
- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow v$
- $E \rightarrow n$

Geben Sie eine Grammatik G' mit $L(G') = L(G)$ an, die nur Parse-Trees ermöglicht, die $*$ und $+$ links-assoziativ interpretieren, und die "Punktrechnung vor Strichrechnung" respektiert.

Lösung: $G' = \langle V'_N, V_T, P, E \rangle$ mit $V'_N = \{E, T, F\}$ und P wie folgt:

- $E \rightarrow E + T$
- $E \rightarrow T$
- $T \rightarrow T * F$
- $T \rightarrow F$
- $F \rightarrow (E)|v|n$

Fortsetzung

Aufgabe 6 (1+1+1+6 Punkte)

Betrachten Sie das folgende Programm.

- a) Bestimmen Sie die Ausgabe des Programms (also den Rückgabewert von `log2()`) für folgende Eingabewerte:
- i) 2
 - ii) 7
 - iii) 9
- b) Wie können Sie die Funktion `log2()` optimieren, ohne ihr Ergebnis zu verändern? Geben Sie die Optimierungen und die endgültige Funktion an.

```
Integer pow(Integer base, Integer exp)
{
    Integer res;
    res = 1;
    while(exp != 0)
    {
        res = res*base;
        exp = exp - 1;
    }
    return res;
}
```

```
Integer log2(Integer n)
{
    Integer guess;

    guess = n;

    while(0<=pow(2, guess))
    {
        if(pow(2, guess)<=n)
        {
            if(pow(2, guess+1) > n)
            {
                return guess;
            }
        }
        if(pow(2, guess)>n)
        {
            guess = guess/2;
        }
        else
        {
            if(pow(2, guess)<n)
            {
                guess = guess+1;
            }
        }
    }
    print guess;
    print "\n";
    return 0;
}
```

Fortsetzung

Lösung:

- a) $2 \rightarrow 1; 7 \rightarrow 2; 9 \rightarrow 3$
- b) – Common subexpression elimination: `pow(2, guess)`
 – While-Bedingung ist immer wahr: Ersetze durch `0<=1`
 – Code nach der While-Schleife ist nie erreichbar, streichen

```
Integer pow(Integer base, Integer exp)
```

```
{
    Integer res;
    res = 1;
    while(exp != 0)
    {
        res = res*base;
        exp = exp - 1;
    }
    return res;
}
```

```
Integer log2(Integer n)
```

```
{
    Integer guess;

    guess = n;

    while(0<=1)
    {
        Integer pow2;
        pow2=pow(2, guess);
        if(pow2<=n)
        {
            if(pow(2, guess+1) > n)
            {
                return guess;
            }
        }
        if(pow2>n)
        {
            guess = guess/2;
        }
        else
        {
            if(pow2<n)
            {
                guess = guess+1;
            }
        }
    }
    return 0;
}
```