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

Aufgabe 2 (10 Punkte)
Das folgende *nanoLang*-Programm enthält 5 Typen von Fehlern. Markieren Sie mindestes 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;
}
```

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**
`char -> int`	`int()`
`<int, <int, char>>`	`<char>`
`(int -> (int))`	`real->hans`
`(int -> (int -> int))`	`char,int`
`<int -> char, char>`	`<int -> char<`

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.

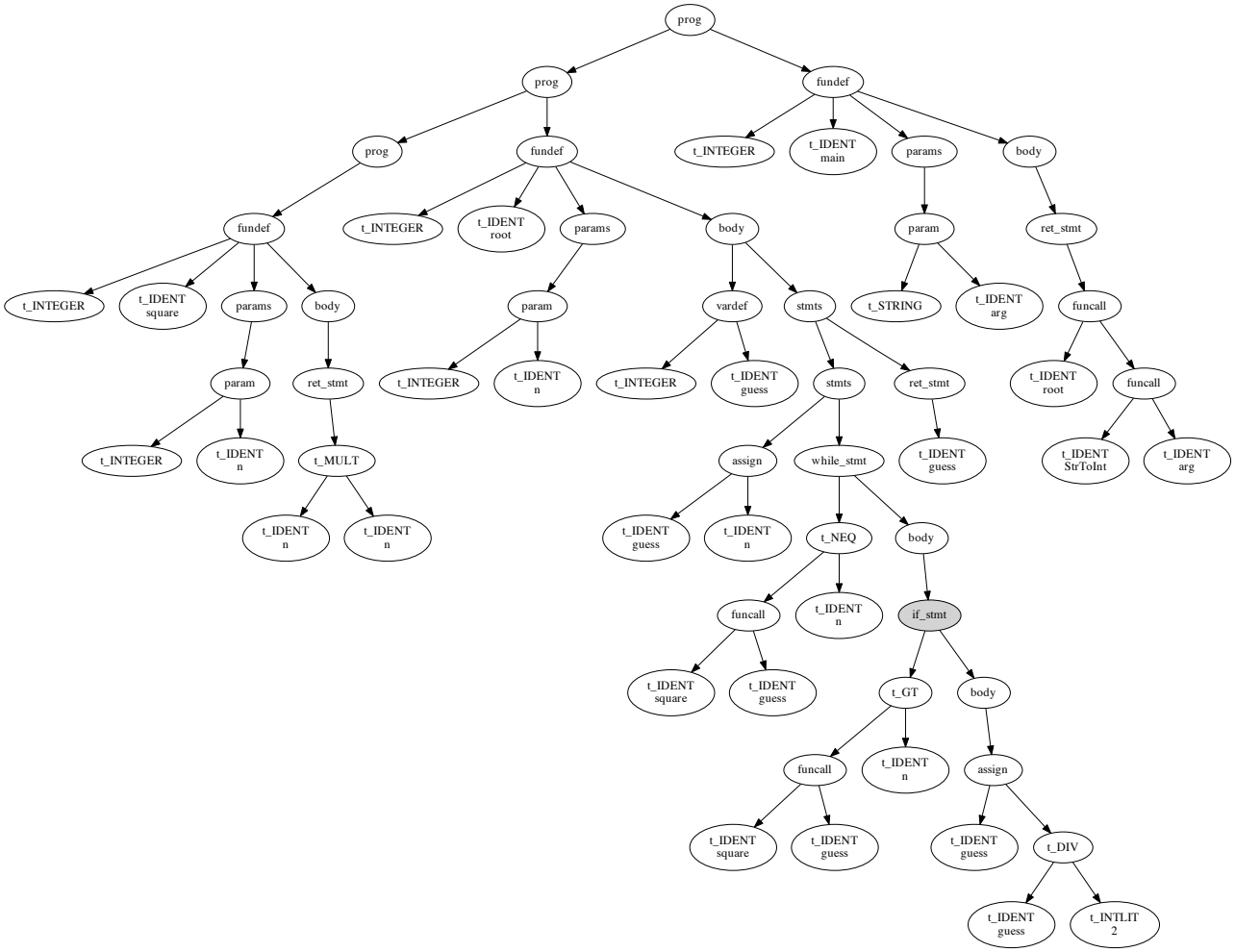
1. `char -> int`
2. `<int,char->int> -> char -> char`

Fortsetzung

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 geparst 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.


prog
prog
fundef
prog
fundef
t_INTEGER
t_IDENT main
params
body
fundef
t_INTEGER
t_IDENT root
params
body
param
ret_stmt
t_INTEGER
t_IDENT square
params
body
param
param
vardef
stmts
t_STRING
t_IDENT arg
funcall
param
ret_stmt
t_INTEGER
t_IDENT n
t_INTEGER
t_IDENT guess
stmts
ret_stmt
t_IDENT root
funcall
t_INTEGER
t_IDENT n
t_MULT
assign
while_stmt
t_IDENT guess
t_IDENT StrToInt
t_IDENT arg
t_IDENT n
t_IDENT n
t_IDENT guess
t_IDENT n
t_NEQ
body
funcall
t_IDENT n
if_stmt
t_IDENT square
t_IDENT guess
t_GT
body
funcall
t_IDENT n
assign
t_IDENT square
t_IDENT guess
t_IDENT guess
t_DIV
t_IDENT guess
t_INTLIT 2

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:

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

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.

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

– Ende der Klausur –