Einführungsveranstaltung: Compilerbau-Praktikum

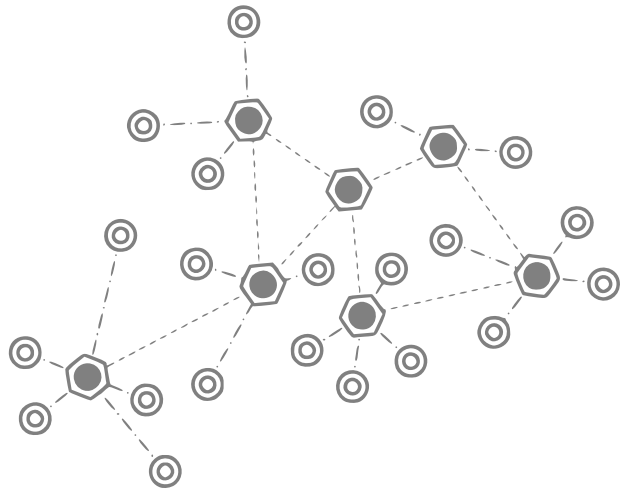
André-Marcel Hellmund

amh@hp.com

21. Februar 2012

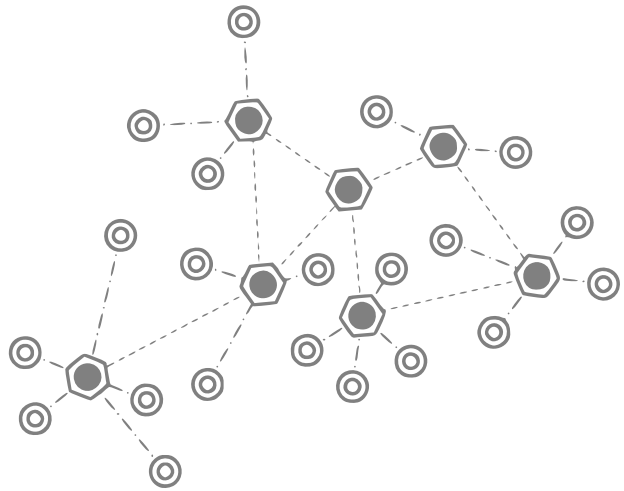
Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



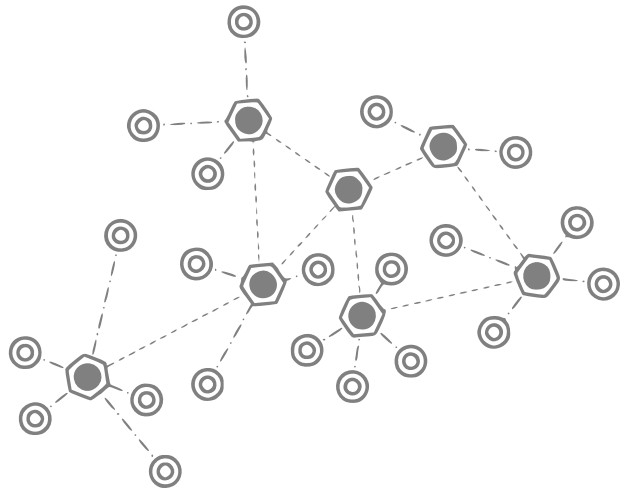
Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken

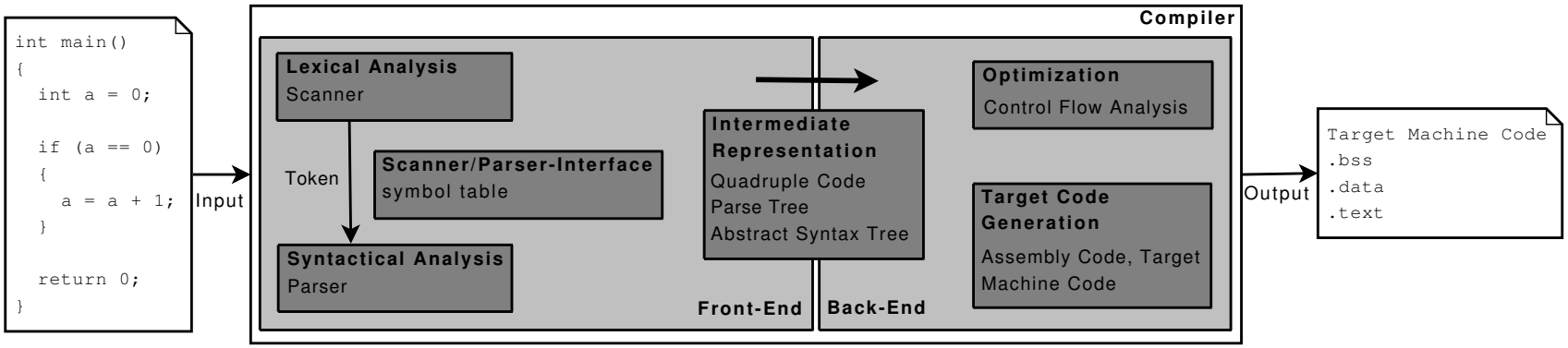


Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



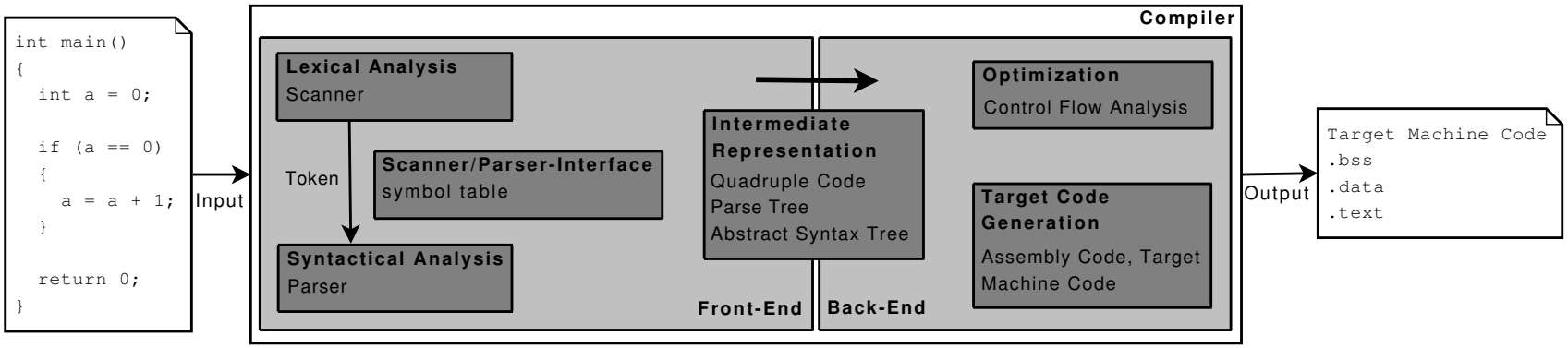
Übersicht



Aufgaben

- Scanner und Parser
- Symboltabelle und Typsystem
- Zwischendarstellung mittels Quadrupel-Code
- Maschinencode

Übersicht

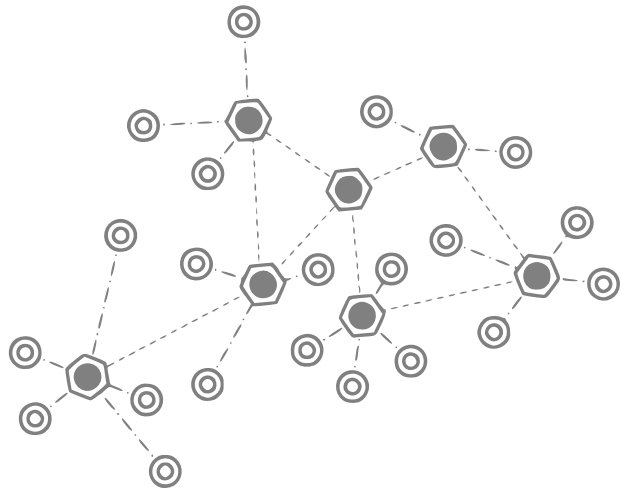


Aufgaben

- Scanner und Parser
- Symboltabelle und Typsystem
- Zwischendarstellung mittels Quadrupel-Code
- Maschinencode

Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Datentypen, Variablen und Funktionen

Datentypen

- Standard-Datentypen: *int* und *void*
- Erweiterte Datentypen: eindimensionale, statische Arrays

Variablen

- Globale und lokale Variablen
- In Unterblöcken (z.B. Schleifenrumpf) sind lokale Variablen

Datentypen, Variablen und Funktionen

Datentypen

- Standard-Datentypen: *int* und *void*
- Erweiterte Datentypen: eindimensionale, statische Arrays

Variablen

- Globale und lokale Variablen
- In Unterblöcken (z.B. Schleifenrumpf) sind lokale Variablen

Funktionen

- Deklarationen (Prototypen) und Definitionen von Funktionen
- mit Rückgabewert und Übergabeparametern

Datentypen, Variablen und Funktionen

Datentypen

- Standard-Datentypen: *int* und *void*
- Erweiterte Datentypen: eindimensionale, statische Arrays

Variablen

- Globale und lokale Variablen
- In Unterblöcken (z.B. Schleifenrumpf) sind lokale Variablen

Funktionen

- Deklarationen (Prototypen) und Definitionen von Funktionen
- mit Rückgabewert und Übergabeparametern

Binäre Operationen in Ausdrücken

Binäre arithmetische Operatoren

- Addition ($+$), Subtraktion ($-$), Multiplikation ($*$), Shift-Left ($<<$), Shift-Right ($>>$)

Binäre Vergleichsoperatoren

- größer ($>$), größer gleich ($>=$), kleiner ($<$), kleiner gleich ($<=$), gleich ($==$), ungleich ($!=$)

Binäre Operationen in Ausdrücken

Binäre arithmetische Operatoren

- Addition ($+$), Subtraktion ($-$), Multiplikation ($*$), Shift-Left ($<<$), Shift-Right ($>>$)

Binäre Vergleichsoperatoren

- größer ($>$), größer gleich ($>=$), kleiner ($<$), kleiner gleich ($<=$), gleich ($==$), ungleich ($!=$)

Binäre logische Operatoren

- UND (&&), ODER (||)

Binäre Operationen in Ausdrücken

Binäre arithmetische Operatoren

- Addition ($+$), Subtraktion ($-$), Multiplikation ($*$), Shift-Left ($<<$), Shift-Right ($>>$)

Binäre Vergleichsoperatoren

- größer ($>$), größer gleich ($>=$), kleiner ($<$), kleiner gleich ($<=$), gleich ($==$), ungleich ($!=$)

Binäre logische Operatoren

- UND ($\&\&$), ODER ($||$)

Unäre Operationen und Klammerung in Ausdrücken

Unäre arithmetische Operatoren

- Minus ($-$)

Unäre logische Operatoren

- NICHT (!)

Unäre Operationen und Klammerung in Ausdrücken

Unäre arithmetische Operatoren

- Minus ($-$)

Unäre logische Operatoren

- NICHT (!)

Unäre Feldoperatoren

- Array-Zugriff([])

Unäre Operationen und Klammerung in Ausdrücken

Unäre arithmetische Operatoren

- Minus ($-$)

Unäre logische Operatoren

- NICHT (!)

Unäre Feldoperatoren

- Array-Zugriff([])

Schleifen, Verzweigungen, Funktionsaufrufe

Schleifen

- while, do-while

Verzweigungen

- if-then, if-then-else

Schleifen, Verzweigungen, Funktionsaufrufe

Schleifen

- while, do-while

Verzweigungen

- if-then, if-then-else

Funktionsaufrufe

- ohne variable Anzahl von Übergabeparametern
- Rückgabewerte aus Funktionen (return)

Schleifen, Verzweigungen, Funktionsaufrufe

Schleifen

- while, do-while

Verzweigungen

- if-then, if-then-else

Funktionsaufrufe

- ohne variable Anzahl von Übergabeparametern
- Rückgabewerte aus Funktionen (return)

Alles andere wird nicht unterstützt

Beispielsweise keine explizite Typumwandlung durch eingeklammerte Typen vor einem Variablen-Bezeichner

Schleifen, Verzweigungen, Funktionsaufrufe

Schleifen

- while, do-while

Verzweigungen

- if-then, if-then-else

Funktionsaufrufe

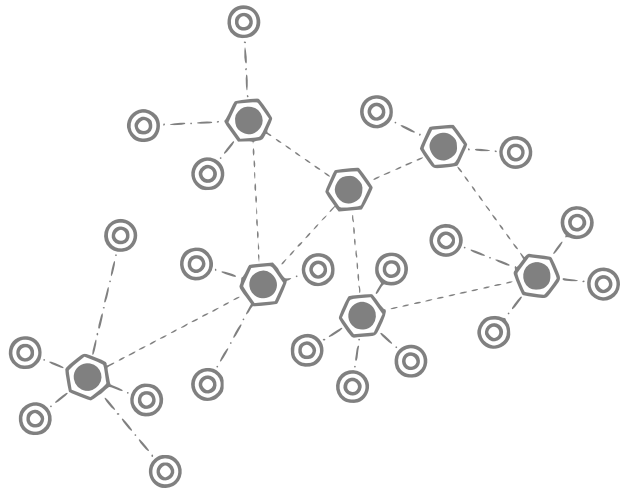
- ohne variable Anzahl von Übergabeparametern
- Rückgabewerte aus Funktionen (return)

Alles andere wird nicht unterstützt

Beispielsweise keine explizite Typumwandlung durch eingeklammerte Typen vor einem Variablen-Bezeichner

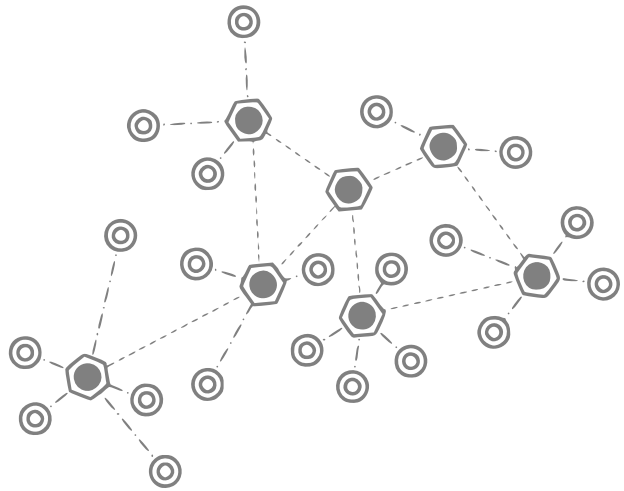
Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Organisatorisches

Gruppenarbeit

- 3-4 Gruppenmitglieder
- 7x 3er Gruppe und 1x 4er Gruppe

Bewertung

- bestanden oder nicht bestanden

Organisatorisches

Gruppenarbeit

- 3-4 Gruppenmitglieder
- 7x 3er Gruppe und 1x 4er Gruppe

Bewertung

- **bestanden** oder **nicht bestanden**

Abgabe

- spätester Abgabetermin ist: **12.06.2012**
- 3 Zwischenabgaben

Organisatorisches

Gruppenarbeit

- 3-4 Gruppenmitglieder
- 7x 3er Gruppe und 1x 4er Gruppe

Bewertung

- **bestanden** oder **nicht bestanden**

Abgabe

- spätester Abgabetermin ist: **12.06.2012**
- 3 Zwischenabgaben

Zeitplan

Start am 21. Februar 2012

- Erste Schritte

Zwischenabgabe nach 4 Wochen, bis zum 20. März 2012

- Scanner und Parser

Zeitplan

Start am 21. Februar 2012

- Erste Schritte

Zwischenabgabe nach 4 Wochen, bis zum 20. März 2012

- Scanner und Parser

Zwischenabgabe nach 4 Wochen, bis zum 17. April 2012

- Symboltabelle und Typsystem

Zeitplan

Start am 21. Februar 2012

- Erste Schritte

Zwischenabgabe nach 4 Wochen, bis zum 20. März 2012

- Scanner und Parser

Zwischenabgabe nach 4 Wochen, bis zum 17. April 2012

- Symboltabelle und Typsystem

Zwischenabgabe nach 4 Wochen, bis zum 15. Mai 2012

- Zwischencode

Zeitplan

Start am 21. Februar 2012

- Erste Schritte

Zwischenabgabe nach 4 Wochen, bis zum 20. März 2012

- Scanner und Parser

Zwischenabgabe nach 4 Wochen, bis zum 17. April 2012

- Symboltabelle und Typsystem

Zwischenabgabe nach 4 Wochen, bis zum 15. Mai 2012

- Zwischencode

Endabgabe nach 4 Wochen, bis zum 12. Juni 2012

- Maschinencode

Zeitplan

Start am 21. Februar 2012

- Erste Schritte

Zwischenabgabe nach 4 Wochen, bis zum 20. März 2012

- Scanner und Parser

Zwischenabgabe nach 4 Wochen, bis zum 17. April 2012

- Symboltabelle und Typsystem

Zwischenabgabe nach 4 Wochen, bis zum 15. Mai 2012

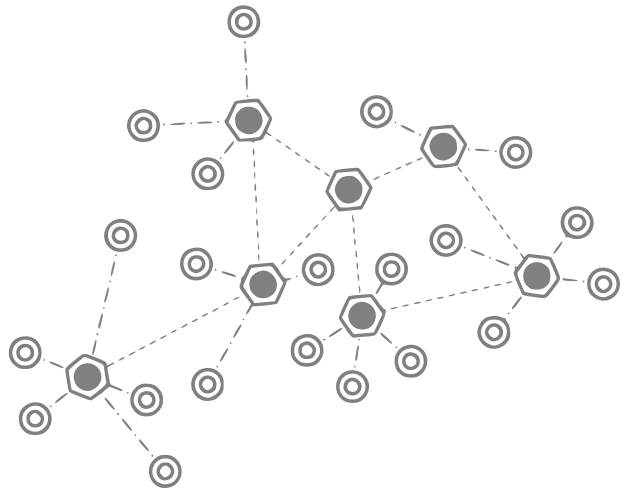
- Zwischencode

Endabgabe nach 4 Wochen, bis zum 12. Juni 2012

- Maschinencode

Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Heute: Erste Schritte

Einarbeiten

- Gruppeneinteilung
- Aufgabenbeschreibung
- Entwicklungstools

Einrichten

- Entwicklungsumgebung

Heute: Erste Schritte

Einarbeiten

- Gruppeneinteilung
- Aufgabenbeschreibung
- Entwicklungstools

Einrichten

- Entwicklungsumgebung

GNU Flex

- Scannerimplementierung

Heute: Erste Schritte

Einarbeiten

- Gruppeneinteilung
- Aufgabenbeschreibung
- Entwicklungstools

Einrichten

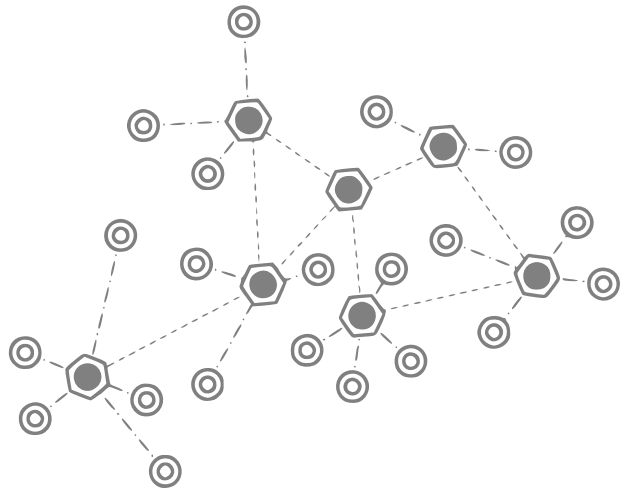
- Entwicklungsumgebung

GNU Flex

- Scannerimplementierung

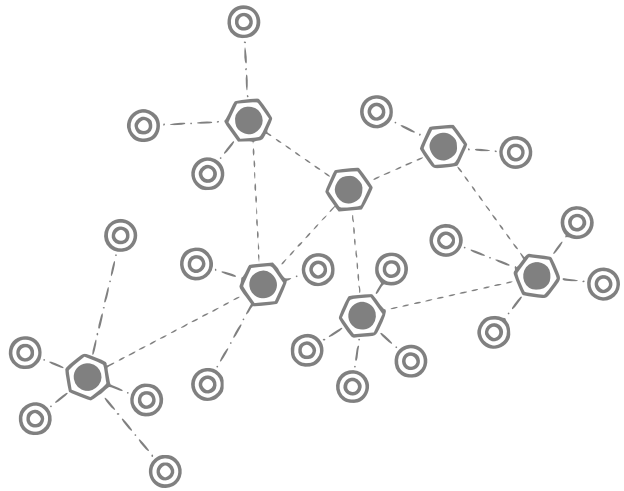
Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Übersicht

Entwicklungsumgebung

- Code-Verwaltung mit SVN (optional)
- Entwicklung mit Eclipse (optional)
- Compilieren mit GCC (Pflicht), auch in Eclipse integriert
- Debuggen mit GDB (Pflicht), auch in Eclipse integriert
- Debuggen mit Flex und Bison (Pflicht)
- Projektbau mit Make (Pflicht), auch in Eclipse integriert

Qualitätssicherung

- Code-Dokumentation mit Doxygen
- Vermeidung von Fehlern im dynamischen Speicher mit Valgrind

Übersicht

Entwicklungsumgebung

- Code-Verwaltung mit SVN (optional)
- Entwicklung mit Eclipse (optional)
- Compilieren mit GCC (Pflicht), auch in Eclipse integriert
- Debuggen mit GDB (Pflicht), auch in Eclipse integriert
- Debuggen mit Flex und Bison (Pflicht)
- Projektbau mit Make (Pflicht), auch in Eclipse integriert

Qualitätssicherung

- Code-Dokumentation mit Doxygen
- Vermeidung von Fehlern im dynamischen Speicher mit Valgrind

Doxygen

Dokumentation von ...

- Dateien, Funktionen, Variablen, Datenstrukturen, ...

Beispiel für eine Variable

```
/** \brief Einzeilige Kurzbeschreibung
 *
 * Mehrzeilige und detaillierte Beschreibung
 */
int numberOfSomething = 0;
```

Doxygen

Dokumentation von ...

- Dateien, Funktionen, Variablen, Datenstrukturen, ...

Beispiel für eine Variable

```
/** \brief Einzeilige Kurzbeschreibung
 *
 * Mehrzeilige und detaillierte Beschreibung
 */
int numberOfSomething = 0;
```

Valgrind

Überblick

- Framework for building dynamic analysis tools
- "There are tools that can automatically detect many memory management and threading bugs, and profile your programs in detail"

Beispiel

```
// Perform a memory leak analysis and show a
// detailed report on lost (directly and
// indirectly) memery
valgrind --tool=memcheck --leak-check=yes
--show-reachable=yes ./programUnderTest
```

Valgrind

Überblick

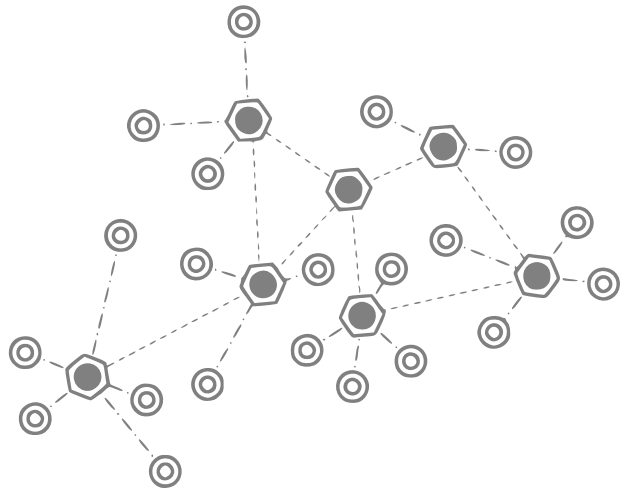
- Framework for building dynamic analysis tools
- "There are tools that can automatically detect many memory management and threading bugs, and profile your programs in detail"

Beispiel

```
// Perform a memory leak analysis and show a
// detailed report on lost (directly and
// indirectly) memery
valgrind --tool=memcheck --leak-check=yes
--show-reachable=yes ./programUnderTest
```

Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Übersicht

GNU Flex: Scanner-Generator

- "Generates programs that perform pattern-matching on text."
- Generator-Programm für Scanner
- Generiert deterministische endliche Automaten

GNU Bison: Parser-Generator

- Generator-Programm für Parser
- Generiert Shift/Reduce-Parser, der eine Rechtsableitung einer LALR(1)-Grammatik (Untermenge kontext-freier Grammatiken) durchführt.
- Generiert endliche Keller-Automaten

Übersicht

GNU Flex: Scanner-Generator

- "Generates programs that perform pattern-matching on text."
- Generator-Programm für Scanner
- Generiert deterministische endliche Automaten

GNU Bison: Parser-Generator

- Generator-Programm für Parser
- Generiert Shift/Reduce-Parser, der eine Rechtsableitung einer LALR(1)-Grammatik (Untermenge kontext-freier Grammatiken) durchführt.
- Generiert endliche Keller-Automaten

GNU Flex und Bison - Dateistruktur

Aufbau

```
Definitionen und Optionen
%%
Regeln
%%
benutzerdefinierter C-Code
```

GNU Flex - Semantik I

Reguläre Ausdrücke

- *
- +
- ?
- \
- []
- ^
- ()
- .
- |

GNU Flex - Semantik II

Flex-spezifische Variablen

- **yyin** - die Eingabedatei (als FILE*)
- **yytext** - enthält die gefundene Zeichenkette
- **yyleng** - die Länge der Zeichenkette
- **yylineno** - die aktuelle Zeilennummer

Flex-spezifische Funktionen

- yylex - die Scanner-Funktion, liefert Tokencode

GNU Flex - Semantik II

Flex-spezifische Variablen

- **yyin** - die Eingabedatei (als FILE*)
- **yytext** - enthält die gefundene Zeichenkette
- **yyleng** - die Länge der Zeichenkette
- **yylineno** - die aktuelle Zeilennummer

Flex-spezifische Funktionen

- **yylex** - die Scanner-Funktion, liefert Tokencode

Beispiel

Aufgabe

Parsing der Sprache: $a^n b^n$ mit $n \geq 1$

Beispiel II

Lexikalische Analyse mit Flex

```
%{
        #include "parser.h"
%}

charb b
%%

a                   { return A; }
{charb}             { return B; }
\n                  { return 0; }
```

Achtung: Die Reihenfolge der Regeln im Regelabschnitt ist wichtig!

Beispiel II

Lexikalische Analyse mit Flex

```
%{
        #include "parser.h"
%}

charb b
%%

a               { return A; }
{charb}         { return B; }
\n              { return 0; }
```

Achtung: Die Reihenfolge der Regeln im Regelabschnitt ist wichtig!

Beispiel III

Syntaktische Analyse mit Bison

```
%start exp
%token A B

%%

exp: A exp B
   | A B
   ;

%%
```

Flex-Tipps I

Scanner: Zeilen- und Spalteninformationen

```
// Automatically track line numbers
%option yylineno

// Track line and column numbers
#define YY_USER_ACTION { \
yylloc->first_line = yylineno; \
yylloc->last_line = yylineno; \
yylloc->first_column = yycolumn; \
yylloc->last_column = yycolumn + yyleng - 1; \
yycolumn += yyleng; }
```

Flex-Tipps II

Scanner: Bison-Kompatibilität

```
// Bison compatibility in API and line and
// column numbers
%option bison-locations
```

Debugging

```
// Scanner: Error on unmatched input
%option warn nodefault

// Flex: report on the scanner generation
%option verbose
```

Flex-Tipps II

Scanner: Bison-Kompatibilität

```
// Bison compatibility in API and line and
// column numbers
%option bison-locations
```

Debugging

```
// Scanner: Error on unmatched input
%option warn nodefault

// Flex: report on the scanner generation
%option verbose
```

Bison-Tipps I

Parser: Zeilen- und Spalteninformationen

```
// Track locations of symbols
%locations
```

Parser: Datentypen auf dem Keller

```
// Semantic values on the stack
%union {
char *pcLiteral;
int iLiteral;
float fLiteral;
...
}
```

Bison-Tipps I

Parser: Zeilen- und Spalteninformationen

```
// Track locations of symbols
%locations
```

Parser: Datentypen auf dem Keller

```
// Semantic values on the stack
%union {
char *pcLiteral;
int iLiteral;
float fLiteral;
...
}
```

Bison-Tipps II

Bison: Debugging

```
// Report on the parser generation
%verbose

// Report when parser generation fails
%error-verbose

// Expect one shift/reduce error
%expect 1
```

Bison-Tipps III

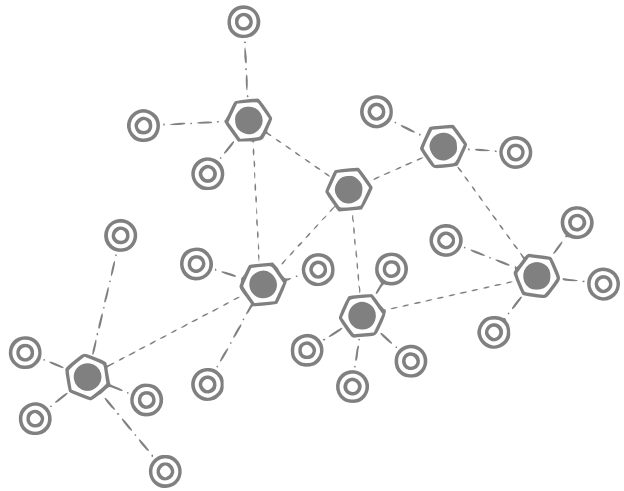
Parser: Debugging

```
// Instrument parser for debugging
%debug

// Enable debugging instrumentation
%initial-action
{
yydebug = 1;
};
```

Agenda

1. Aufgabe
   - Implementierung
   - Sprachumfang
2. Ablauf
   - Organisation
   - Heute: Erste Schritte
3. Tools
   - Nicht-Funktionale Tools
   - Funktionale Tools
   - Bibliotheken



Übersicht

Motivation

- Keine Räder neuerfinden!

Beispiele

- utlist: verkettete Listen
- uthash: Hash-Tabellen

Übersicht

Motivation

- Keine Räder neuerfinden!

Beispiele

- utlist: verkettete Listen
- uthash: Hash-Tabellen

utlist

utlist: verkettete Listen

- http://uthash.sourceforge.net/utlist.html
- "A set of general-purpose linked list macros for C structures [...]."
- "Singly-linked lists, doubly-linked lists, and circular, doubly-linked lists"
- Operationen: PREPEND, APPEND, CONCAT, DELETE, SORT, FOREACH, SEARCH
- Könnte für die Symboltabelle, den Zwischencode oder den Maschinencode nützlich sein

uthash

uthash: Hash-Tabellen

- http://uthash.sourceforge.net/
- "A set of general-purpose hash table macros for C structures [...]".
- Operationen: add, find, delete, count, iterate und sort
- Könnte für die Symboltabelle nützlich sein

Q & A

Fragen?