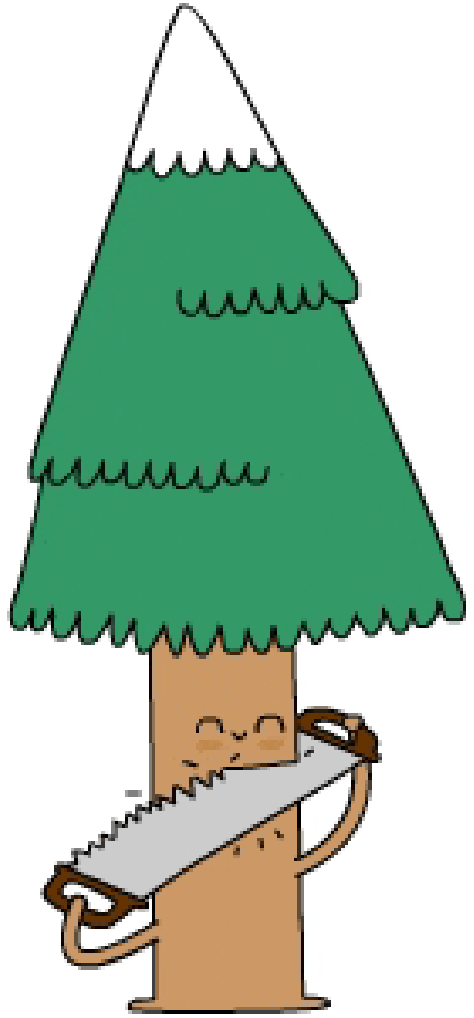
Übung 07.2

Debugging and Testing

0. Organisatorisches

- Über Weihnachten **kein neues** Übungsblatt
- Uni fängt am 02.01.17 wieder an
- Nächstes Übungsblatt:
  - Ausgabe: Do. 05.01.17
  - Abgabe: Do. 12.01.17
- Nächste Übung: Fr. 13.01.17
- **LADET EUCH ALLE VORLESUNGS-SCRIPTE BIS 01.01.17 HERUNTER!!!**

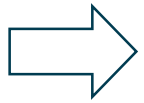


http://giphy.com/gifs/csak-christmas-suicide-csak-D5LC1OORr5bt72

01. Debugging

- 1. Fehler

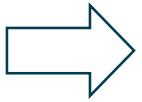
```
68 marker = 0;  // reset the marker
69 temp = marker;
70 marker = marker->next();
```



```
72 // FIXME 1 - immediate segfault, not needed at all
73 // marker = 0;  // reset the marker
74 temp = marker;
75 marker = marker->next();
```

- 2. Fehler

```
62 temp->next (marker->next());
63 delete temp;
64 temp = 0;
65 return 0;
```

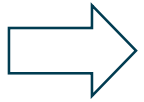


```
65 // FIXME 2 - should be delete marker
66 // delete temp;
67 delete marker;
68 temp = 0;
69 return 0;
```

01. Debugging

- 3. Fehler

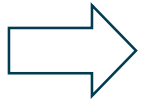
```
56 head_ = new Item<T>(marker->value(),
marker->next());
57 delete marker;
58 marker = 0;
```



```
56 // FIXME 3 - no need to create new
item, just let head_ = marker->next
57 // head_ = new Item<T>(marker-
>value(), marker->next());
58 head_ = marker -> next();
59 delete marker;
60 marker = 0;
```

- 4. Fehler

```
79 do {
80   cout << marker->value() << endl;
81   marker = marker->next();
82} while (marker != 0);
```



```
84 // FIXME 4 - fails when list is empty
...
90 while (marker != 0) {
91   Item<T> *temp = marker;
92   delete marker;
93   marker = temp->next();
94}
```

02. Begriffsdefinitionen: Validierung vs. Verifikation

- **Software Validierung**
  - Erstellen wir das korrekte Produkt?
  - z.B. Kunde in Entwicklung einbinden, Prototyping, Akzeptanztests
  - Prozess der Beurteilung eines Systems oder einer Komponente während oder am Ende des Entwicklungsprozesses, mit dem Ziel festzustellen, ob die spezifizierten Anforderungen erfüllt wurden (nach IEEE 610.12)

- **Software Verifikation**
  - Erstellen wir das Produkt korrekt?
  - z.B. (Junit)-Testing, Beweis, Model Checking
  - Prozess der Beurteilung eines Systems oder einer Komponente mit dem Ziel, ob die Resultate einer gegebenen Entwicklungsphase den Vorgaben der Phase entsprechen (nach IEEE 610.12)

02. Begriffsdefinitionen: Failure vs. Fault

- **Software Failure**
  - Unfähigkeit eines Systems oder einer Komponente ihre geforderte Funktion innerhalb ihres spezifizierten Leistungsumfangs zu erfüllen (nach IEEE 610.12)
  - z.B. Folge der Ausführung eines Programms das „Bugs“ enthält

- **Software Fault**
  - inkorrekter Schritt, Prozess oder inkorrekte Datendefinition in einem Computer Programm (nach IEEE 610.12)
  - Synonym: „Bug im Programm Code“

02. Begriffsdefinitionen: Regression Testing

- erneuter Test bereits getesteter Software
- muss jeweils nach Modifikation des Systems erfolgen, z.B. nach ...
  - … jedem check-in
  - … jeder Dateiänderung
- idealerweise vollautomatisch
- Umsetzung durch xUnit-Frameworks, z.B. JUnit, PHPUnit, etc.

➢ Nachweis, dass keine **neuen** Defekte eingebaut wurden

➢ es funktioniert mindestens all das was zuvor bereits funktioniert hat

02. Begriffsdefinitionen: Testing vs. Model Checking

Testing

- ✓ einfach
- ✓ „billig“
- ✗ unvollständig
- ➢ einfacher tool support
- ➢ allgemein einsetzbar
- ➢ findet Fehler, aber keine Garantie bzgl. Korrektheit

Model Checking

- ✗ komplex
- ✗ „teuer“
- ✓ vollständig
- ➢ Expertentools
- ➢ Hochsicherheitsbereich
- ➢ Beweis für Korrektheit oder Fehlerpfad

02. Begriffsdefinitionen: Gefahr bei nur unit-tests?

Ja, reiner Bottom-Up Ansatz - integrationstest außer Acht gelassen - Systemkomposition kann scheitern!

Top-Down Integration Test

- zuerst Test der Top-Level Module
- Abhängigkeiten zu Low-Level Modulen werden vorerst als Stubs realisiert

✓ Designfehler früh identifizierbar

✓ kein Testtreiber nötig

✗ Stubs müssen realisiert werden

Bottom-Up Integration Test

- zuerst Test der Low-Level Module (Unit-Tests)
- Hinzufügen der höher gelagerten Module erfolgt im Laufe der Zeit

✓ Komponenten früh ausgreift

✓ keine Stubs nötig

✗ Testtreiber nötig

03. Unit-Testing

- Testing über Eclipse und IntelliJ sollte ohne große Probleme laufen

- Commandline-Workaround (Linux):
  - `Interval.java` und `IntervalTest.java` befinden sich im gleichen Ordner

```
1. sudo apt-get install junit4
2. export CLASSPATH=/usr/share/java/junit4-4.12.jar
3. javac *.java
4. java -cp .:/usr/share/java/junit4-4.12.jar
   org.junit.runner.JUnitCore IntervalTest
```

http://www.hwk-chemnitz.de/fileadmin/user_upload/ausbildung/Pruefungen/Gesellenpruefungen/Fragezeichen.JPG

I wish you a merry christmas and a happy new year!