Informatik II
Tutorial 4
Mihai Bâce
mihai.bace@inf.ethz.ch
22.03.2017
Overview

- Debriefing Exercise 3
- Briefing Exercise 4
a) What is the loop invariant for this code

One solution:

```
static int f(int i, int j) {
    int u = i;
    int z = 0;
    while (u > 0) {
        z = z + j;
        u = u - 1;
    }
    return z;
}
```

Term will always be 0
b) Proove the correctnes of the code with the loop invariant.

```c
static int f(int i, int j) {
    int u = i;
    int z = 0;
    while (u > 0) {
        z = z + j;
        u = u - 1;
    }
    return z;
}
```

- $z = 0 \rightarrow u \times j = i \times j$
- $z = n \times j, \quad u = i - n, \quad \text{inv} = 0$
- $z = (n+1) \times j, \quad u = i - (n+1), \quad \text{inv} = 0$
- $u = 0 \rightarrow z = i \times j$
c) what if line 5 and 6 are changed to $z = z$; and $u = u$?

The loop invariant is still valid

$$z + u \times j - i \times j$$

```java
static int f(int i, int j) {
    int u = i;
    int z = 0;
    while (u > 0) {
        z = z
        u = u
    }
    return z;
}
```
c) what if line 5 and 6 are changed to \( z = z \); and \( u = u \)?

Using Hoare logic:

If condition and invariant
Are true before
The loop body

\[ \{ C \land I \} \quad \text{body} \quad \{ I \} \]

\[ \{ I \} \quad \text{while (C) body} \quad \{ \neg C \land I \} \]

then the negated condition
(u=0) holds after the loop body

But does it mean that the implementation is correct?
U3.A1 Program verification

c) what if line 5 and 6 are changed to z=z; and u=u?

No, the proofs so far only show **partial correctness**
For **total correctness**, we also need to prove termination

Does the program terminate?
NO

Only **partially correct**

```c
static int f(int i, int j) {
    int u = i;
    int z = 0;
    while (u > 0) {
        z = z
        u = u
    }
    return z;
}
```

u is always > 0

1. Objects and references (e.g. Strings)
   - String vs. StringBuffer
   - Caesar cypher
   - Encrypt and decrypt, understand how the program works
U3.A2 Decrypt

- Inverse of encrypt
- Take each character and subtract 3 from its ASCII code

- How do you access each character? \( s\text{.charAt}(index) \)

```java
/**
 * Decrypts input text based on the CaesarChiffre (i.e., removing 3 from the
 * ASCII code of each character). The decryption employs StringBuffers
 * (instead of Strings).
 *
 * @param s
 * ciphertext to be decrypted
 *
 * @return String
 */
public static String decrypt(String s) {
    StringBuffer ret = new StringBuffer();
    for (int i = 0; i != s.length(); ++i) {
        ret.append((char) (s.charAt(i) - 3));
    }
    return ret.toString();
}
```
<table>
<thead>
<tr>
<th>ASCII Character Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>
U3.A2 Main

- What is different?
  - Encrypt is much slower than decrypt. Why?
  - StringBuffer is more efficient for appending
  - Strings are immutable
    - Any modification leads to a new copy of the object.
U3.A2 Strings

- Why use Strings in the first place?
  - Strings are constants and allow for optimizations
  - Strings are immutable, which could be a requirement in some cases
  - The biggest benefit for StringBuffer is when we append/modify the string at runtime
2a) Clause

<table>
<thead>
<tr>
<th>Expression</th>
<th>Possible</th>
<th>Impossible</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_2$</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>$(\sim X_1)$</td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td>$(\sim (X_1 \text{ OR } \sim X_2))$</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>$(X_2 \text{ OR } (\sim X_1 \text{ OR } X_2))$</td>
<td>O</td>
<td>X</td>
</tr>
</tbody>
</table>
U3.A3 Syntax diagrams

Var: \( X_1, \ldots, X_n \)  
Clause: \((\text{Clause})\)  
Expr: \( (\text{Clause}) \rightarrow \text{AND} \rightarrow (\text{Clause}) \)

2b) Expr

<table>
<thead>
<tr>
<th>Possible</th>
<th>Impossible</th>
</tr>
</thead>
<tbody>
<tr>
<td>((X_1 \text{ OR } X_2) \text{ AND } (\sim X_1))</td>
<td>✗</td>
</tr>
<tr>
<td>((X_1) \text{ AND } (\sim X_1 \text{ OR } \sim X_2) \text{ AND } (X_2))</td>
<td>✗</td>
</tr>
</tbody>
</table>
How do we change it to allow empty trees and successors?
private static int parseTree(String kd, int offset) throws ParseException {
    if (offset >= kd.length()) {
        throw new ParseException("Unexpected end of string", offset);
    }
    if (kd.charAt(offset) == '-') {
        return offset + 1;
    }
    else {
        offset = parseNode(kd, offset);
        if ((offset < kd.length()) && (kd.charAt(offset) == '(')) {
            offset += 1;
            offset = parseSubtree(kd, offset);
            if ((offset < kd.length()) && (kd.charAt(offset) == ')')) {
                offset += 1;
            }
        }
        else {
            throw new ParseException("expected ")", offset);
        }
    }

    return offset;
}
private static int parseSubtree(String kd, int offset) throws ParseException {
    if (offset >= kd.length()) {
        throw new ParseException("unexpected end of string after ", offset);
    }
    offset = parseTree(kd, offset);
    while ((offset < kd.length()) && (kd.charAt(offset) == ',')) {
        offset += 1;
        offset = parseTree(kd, offset);
    }
    return offset;
}
U3.A4 Syntax checker

```
private static int parseNode(String kd, int offset) throws ParseException {
    if (offset >= kd.length()) {
        throw new ParseException("Expected a node", offset);
    }

    if (Character.isUpperCase(kd.charAt(offset))) {
        return offset + 1;
    } else {
        throw new ParseException(String.format("'%c' is not a valid node name", kd.charAt(offset)), offset);
    }
}
```
public static void parse(String kd) throws ParseException {
    int offset = parseTree(kd, 0);
    if (offset != kd.length()) {
        throw new ParseException("Garbage at the end of the tree", offset);
    }
}
Overview

- Debriefing Exercise 3
- Briefing Exercise 4
Stack

- Abstract data type
- Collection of elements
- LIFO principle
  - Last in, first out
- Two main operations: Push and Pop
U4.A1

- Constructor
  - Initializes internal Array
  - Capacity is an argument to the constructor

- `toString()` with `StringBuffer`
  - Expected Output: "[e0, e1, e2, ...]"
  - Concatenation
    - String: `str += "bar";`
    - StringBuffer: `buf.append("bar");`

- `grow()`
  - Capacity doubled, copy old values
U4.A1

- push(), pop(), peek(), empty()
  - Standard stack functions
  - Arguments are of type int
  - If necessary, call grow()

- size()
  - Number of elements currently on the stack

- capacity()
  - Total number of elements which fit on the current stack until the next grow
Ackermann function

- Recursive Definition

\[ A(0, m) = m + 1 \]
\[ A(n + 1, 0) = A(n, 1) \]
\[ A(n + 1, m + 1) = A(n, A(n + 1, m)) \]

- Grows extremely fast

  - \( A(3, 3) = 61 \)
  - \( A(4, 2) \) has already 19729 decimal places!!
U4.A2

- A(1,1) given as example in the homework

- Calculate A(2,1) by hand
  - A(2,1) = A(1+1, 0+1) = A(1, A(2,0)) …

- Write down all the steps!
\[ A(4, 3) = A(3, A(4, 2)) \]
\[ = A(3, A(3, 3)) \]
\[ = A(3, A(3, 2)) \]
\[ = A(3, A(3, 1)) \]
\[ = A(3, A(3, 0)) \]
\[ = A(3, A(3, 2^0)) \]
\[ = A(3, A(3, 2^{2^0})) \]
\[ = A(3, A(3, 2)) \]
\[ = A(3, A(3, 2^{2}) - 3) \]
\[ = 2^{2^{165536}} - 3. \]
U4.A2

- Specify the algorithm using the usual two stack operations:
  - push(x)
  - x = pop()

- Pseudocode:
  - No language-specific syntax
  - Pseudocode is self-explanatory
  - Based on comments

- The function has the property that one can not say in advance how deep the recursion is
  - Use while instead of for-loop!
U4.A2 Iterative approach

- Ackermann’s formula always requires (exactly) two values:
  - The currently required values should be at the top of the stack…
  - What does it mean when there is one item left in the stack?

```java
Stack stack = new Stack();
stack.push(4);
stack.push(7);

while(stack.size() != 1)
{
  . . .
}
```

stack
U4.A2 Implementation

If \( n = 0 \) → \( \text{result} = m + 1 \)

else if \( m = 0 \) → push(n-1), push(1)

else push(n-1), push(n), push(m-1)
U2.A2

Start
A(1,1)

Iteration
size >= 2?
m = 0
n = 0

n == 0?
m + 1
n - 1

m == 0?
1

else
n
m - 1

End
A(1,1) = 3

---

A(1,1)
A(1,0)
A(0, 1)
<- 2
<- 3
A(0, 2)
<- 2
<- 3

Stack

m = 0
m = 0
m = 0

By Leyna Sadamori
U4.A2 Hints

- **Stack**
  - The stack from U4.A1
  - The interface should NOT be modified

- “Snapshots”
  - With toString() method of the stack

- I cannot do U4.A1
  - Use java.util.Stack<Integer>
    - you just need push(), pop(), size und toString()
  - If necessary: send me an Email
U4.A3 Bytecode

- Before you disassemble the code, it must be compiled

- For Linux and Mac users:
  - Use the >> operator in the terminal to send the output to a file
  - E.g.: javap -c RecursiveAckermann >> output.txt
U4.A3 Bytecode

For Windows:

D:\Projects\DisassemblerDemo>
javac JavapTip.java  //compiler
java JavapTip   //run
javap -c -private JavaTip  //disassembler

Common mistake: "javap is not recognized as an internal or external command, operable program or batch file"

Reason: java binaries are not defined in System variable PATH

Solution: RClick on Computer → Properties → Advanced System Settings → Environment Variables → PATH → add (where you installed the Java JDK) → save and restart Windows

;C:\Program Files\Java\jdk1.7.0_31\bin
public int greaterThen(int intOne, int intTwo) {
    if (intOne > intTwo) {
        return 0;
    } else {
        return 1;
    }
}
**U4.A3 Bytecode**

- **Instructions:**
  - iload_n : load int from local variable
  - aload_n : load reference from local variable
  - if_icmp<cond> : Branch if int comparison succeeds
    - E.g. if_icmple: le = less or equal
  - if<cond>: Branch if comparison to zero succeeds
    - Ifeq: equal to 0
    - Ifne: not equal to 0
  - Invokevirtual: invoke instance method

- **Documentation:**
Have Fun!