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

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

One solution:

\[ z + u^*j - i^*j = 0 \text{ and } u \geq 0 \]

Term will always be 0
U3.A1 Program verification

static int f(int i, int j) { assert(i >= 0 && j >= 0);
    int u = i, z = 0;
    // {z + u*j - i*j = 0 and u >= 0} ==> ok 0 + i*j - i*j = 0 and i >= 0
    while (u > 0) {
        // {z + u*j - i*j = 0 and u >= 0 and u > 0}
        // {z + j - j + u*j - i*j = 0 and u > 0}
        z = z + j;
        // {z - j + u*j - i*j = 0 and u > 0}
        // {z - j + (u - 1 + 1)*j - i*j = 0 and u > 0}
        u = u - 1;
        // {(z - j) + (u + 1)*j - i*j = 0 and u + 1 > 0} =>
        // Loop invariant holds because z - j + u*j + j - i*j = z + u*j - i*j = 0
        // {z + u*j - i*j = 0 and u >= 0}
    }
    // {z + u*j - i*j = 0 and u >= 0 and u <= 0}
    return z;
}

Partial Correctness: because u <= 0 and u >= 0
==> u = 0  it follows that z - i * j = 0  =>  z = i*j

Loop invariant:
z + u*j - i*j = 0 and u >= 0
U3.A1 Program verification

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

The loop invariant is still valid

\[ z + u \times j - i \times j \]

```c
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

\[ \{ C \land \bot \} \text{ body} \{ \bot \} \]

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**

U3.A2

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.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
 */
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>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>MUL (null)</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>#32</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>#64</td>
<td>\</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>#33</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>#65</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>#34</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>#66</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>#35</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>#67</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>#36</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>#68</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>#37</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>#69</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>#38</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>#70</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>#39</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>#71</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>#40</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>#72</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>HT (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>#41</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>#73</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>#42</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>#74</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>#43</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>#75</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF (NP form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>#44</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>#76</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>#45</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>#77</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>#46</td>
<td>&gt;</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>#78</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>#47</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>#79</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>#48</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>#80</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>#49</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>#81</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>#50</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>#82</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>#51</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>#83</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>#52</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>#84</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>5</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>#53</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>#85</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>#54</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>#86</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>7</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>#55</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>#87</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td>#56</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>#88</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>#57</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>#89</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>A</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td>#58</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td>#90</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>B</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td>#59</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td>#91</td>
<td>[</td>
</tr>
<tr>
<td>28</td>
<td>C</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td>#60</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td>#92</td>
<td>\</td>
</tr>
<tr>
<td>29</td>
<td>D</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td>#61</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td>#93</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>E</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td>#62</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td>#94</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>F</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td>#63</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td>#95</td>
<td>_</td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com
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></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: $X_1 \rightarrow \sim \rightarrow \cdots \rightarrow X_n$

Clause: $\rightarrow ( \rightarrow \text{Var} \rightarrow )$ OR

Expr: $\rightarrow \text{Clause} \rightarrow \text{AND} \rightarrow \text{Clause}$

2b) Expr

<table>
<thead>
<tr>
<th>Possible</th>
<th>Impossible</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(X_1 \lor X_2) \land \lnot X_1$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$(X_1) \land (\lnot X_1 \lor \lnot X_2) \land (X_2)$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>
U3.A4

- 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;
}
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
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
U4.A2

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!
\[ A(4,3) = A(3, A(4,2)) \]
\[ = A(3, A(3, A(4,1))) \]
\[ = A(3, A(3, A(3, A(4,0)))) \]
\[ = A(3, A(3, A(3, A(3,1)))) \]
\[ = A(3, A(3, A(3, A(2, A(3,0))))) \]
\[ = A(3, A(3, A(3, A(2, A(2,1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(2,0)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(1,1)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(1,0)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(0, A(0,1))))))) \]
\[ = A(3, A(3, A(3, A(2, A(1, A(0,2)))))) \]
\[ = A(3, A(3, A(3, A(2, A(1,3)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(1,2))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(1,1))))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(1,0))))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0, A(0,1))))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0,2)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0, A(0,3)))))) \]
\[ = A(3, A(3, A(3, A(2, A(0,4)))))) \]
\[ = A(3, A(3, A(3, A(2,5)))) \]
\[ = \ldots \]
\[ = A(3, A(3, A(3,13))) \]
\[ = \ldots \]
\[ = A(3, A(3, 65533)) \]
\[ = \ldots \]
\[ = A(3, 2^{65536} - 3) \]
\[ = \ldots \]
\[ = 2^{65536} - 3. \]
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 means when there is one item left in the stack?

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

while(stack.size() != 1)
{
    ...
}
```
U4.A2 Implementation

```
stack.push(m)
stack.push(n)
```

```
if n == 0 → result = m+1
else if m == 0 → push(n-1), push(1)
else push(n-1), push(n), push(m-1)
```
U2.A2

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

Start: A(1,1) → Push

Iteration:

- size >= 2?
  - m = 0
  - n = 0
  - Pop
- n == 0?
  - m + 1
  - Push
- m == 0?
  - n - 1
  - Push
  - 1
  - Push
  - n - 1
  - Push
- else
  - n
  - Push
  - m - 1

End: A(1,1) = 3 → Pop

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\jdkX.Y.Z\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!