36 Organisation of Memory – Summary

J. Kleinöder, D. Lohmann, V. Sieh, P. Wägemann

Lehrstuhl für Informatik 4
Systemsoftware
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Summer Term 2024

http://sys.cs.fau.de/lehre/ss24
Static vs. Dynamic Allocation

- For µC development static allocation is preferred
  - **Advantage:** The required memory is already known during compilation/linking (can be returned with `size/avr-size` command)
  - Problems with memory limits are detected upfront (memory is scarce!)
  - When possible, memory should be allocated with **static** variables
    - Always consider the rule of narrowest scope
    - Always apply the rule of shortest possible “reasonable” lifespan
  - In comparison, a heap is **more expensive** should be avoided
    - Additional costs in memory for management structures and code
    - Memory required during runtime complicated to estimate
    - Risk of memory leaks and programming errors

---

<table>
<thead>
<tr>
<th>text</th>
<th>data</th>
<th>bss</th>
<th>dec</th>
<th>hex</th>
<th>filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>682</td>
<td>10</td>
<td>6</td>
<td>698</td>
<td>2ba</td>
<td>sections.avr</td>
</tr>
</tbody>
</table>

Sizes of the sections of the program

© klsw System-Level Programming (ST 24) 36 Organisation of Memory – Summary – Static vs. Dynamic
When developing for an **operating-system platform** it can be sensible to use **dynamic allocation**

- **Advantage:** dynamic adaption to the size of the input data (e.g., for strings)
- Reduced risk of **buffer-overflow attacks**

If possible, allocate memory for input data on the heap

- Still, the risk of **programming errors and memory leaks** remains!