Data Structures in Memory

Arrays and Pointers
Structs / records / objects
Linked lists

Warning: we’ll start drawing memory in different ways than high addresses up, low addresses down.

Array Layout

Array of data type T and length N
Contiguously allocated memory region of N * (size of T in bytes)

Array of 12 ASCII characters

Array of 5 ints

Array of 3 doubles

Array of 3 addresses

Two-Dimensional Arrays

Declaration
2D array of data type T
R rows, C columns
Type T element requires K bytes

Array layout and size?


Nested Arrays (C)

Declaration
2D array of data type T
R rows, C columns
Type T element requires K bytes

Layout
Row-major ordering

int A[R][C];
Nested Array Access (C)

Each row is an array.

- Each element of type T requires K bytes
- Address of A[R][C] = __________

\[
\begin{array}{c}
A[0][0] \quad A[0][1] \quad \ldots \quad A[0][C-1] \\
A[1][0] \quad A[1][1] \quad \ldots \quad A[1][C-1] \\
\vdots \\
A[R-1][0] \quad A[R-1][1] \quad \ldots \quad A[R-1][C-1] \\
\end{array}
\]

Strange Referencing Examples (C)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sea[3][3]</td>
<td>76+20<em>3+4</em>3 = 148</td>
<td>1 Yes</td>
</tr>
<tr>
<td>sea[2][5]</td>
<td>76+20<em>2+4</em>5 = 136</td>
<td>9 Yes</td>
</tr>
<tr>
<td>sea[2][-1]</td>
<td>76+20<em>2+4</em>-1 = 112</td>
<td>5 Yes</td>
</tr>
<tr>
<td>sea[4][-1]</td>
<td>76+20<em>4+4</em>-1 = 152</td>
<td>5 Yes</td>
</tr>
<tr>
<td>sea[0][19]</td>
<td>76+20<em>0+4</em>19 = 152</td>
<td>5 Yes</td>
</tr>
<tr>
<td>sea[0][-1]</td>
<td>76+20<em>0+4</em>-1 = 72</td>
<td>77 No</td>
</tr>
</tbody>
</table>

C does not do any bounds checking.
Row-major array layout is guaranteed.

Multi-Level Arrays (Java)

```java
int[][] zips = new int[3][];
zips[0] = new int[5] {0, 2, 4, 8, 1};
```

Run-time (Dynamic) Memory Allocation

Memory allocator tracks what memory is available and what memory is being used.

Java: new C: malloc and free

For our purposes:
Procedure allocate:
- \$a0 = number of bytes required
- \$v0 = address of available word-aligned memory block of at least \$a0 bytes

Write MIPS code to implement:
- \$v1 = \$v1 + \$v1 - 1
- \$s1 = \$s1 + \$s1 + 3

null checking and bounds checking.
1. Decide on memory layout for all ListNodes.

2. Implement add:

```java
static void add(ListNode head, ListNode n) {
    ListNode cursor = head;
    while (cursor.next != null) {
        cursor = cursor.next;
    }
    cursor.next = n;
}
```