HAD Debugging

1 HAD Debugging

Boilerplate Code Generation for LAMBDAs with a top-level `LET`.

This task generates precise boilerplate code for Excel 365. The generated code helps debug `LAMBDAs` using the Helper Array Debugging (HAD) technique. We give a `LAMBDA` template and show the corresponding translation, which we call HAD. HAD follows the **Principle of Least Knowledge**: The main functionality in the `INPUT***` does not need to know how the debugging is achieved. The debugging formulas are cleanly separated from the main functionality shown in `OUTPUT***`.

HAD does not allow the `echeck` function to be changed in any way. Use `VALUETOTEXT` as shown below!! Please don't delete commas anywhere in the boilerplate code.

INPUT

```
LAMBDA(a_1,a_2,a_3, ... LET(
  m_1, formula_1,
  m_2, formula_2,
  m_3, formula_3,
  ...
  Show)
)
```

OUTPUT (HAD Instrumented)

```
LAMBDA(a_1,a_2,a_3, ... LET(
  m_1, formula_1,
  m_2, formula_2,
  m_3, formula_3,
  ...
  COMMENT_1, "Debugging Section",
  echeck, LAMBDA(value, IFERROR(value, "ERROR " & VALUETOTEXT(value, 1))),
  Show, IFERROR(HSTACK(
    "a_1", echeck(a_1),
    "a_2", echeck(a_2),
    "a_3", echeck(a_3),
    ...
    "m_1", echeck(m_1),
    "m_2", echeck(m_2),
    "m_3", echeck(m_3),
```

```
    ...
  ), ""),
  Show)
)
```

2 Instrumented Example

Instrument the following `LAMBDA`:

```
=LAMBDA(later,sooner,Table, LET(
  daysEarly, later - sooner,
  luArray, CHOOSECOLS(Table, 1),
  rArray, CHOOSECOLS(Table, 2),
  discount, XLOOKUP(daysEarly, luArray, rArray, "error", -1),
  discount
))(CustomerT[Start Date], CustomerT[Payment Date], DiscountTable)
```

HAD-Instrumented Version

```
=LAMBDA(later, sooner, Table, LET(
  daysEarly, later - sooner,
  luArray, CHOOSECOLS(Table, 1),
  rArray, CHOOSECOLS(Table, 2),
  discount, XLOOKUP(daysEarly, luArray, rArray, "error", -1),

  COMMENT_1, "Debugging Section",
  echeck, LAMBDA(value, IFERROR(value, "ERROR " & VALUETOTEXT(value, 1))),
  Show, IFERROR(HSTACK(
    "later", echeck(later),
    "sooner", echeck(sooner),
    "Table", echeck(Table),
    "daysEarly", echeck(daysEarly),
    "luArray", echeck(luArray),
    "rArray", echeck(rArray),
    "discount", echeck(discount)
  ), ""),
  Show
))(CustomerT[Start Date], CustomerT[Payment Date], DiscountTable)
```

Explanation

- **Original LAMBDA Function:** Computes `daysEarly`, extracts lookup columns, and finds a discount using `XLOOKUP`.
- **Debugging Section:**
  - `COMMENT_1`: Marks the start of the debugging block.
  - `echeck`: Wraps values with error-checking logic and formats errors as readable text.

  - `Show`: Displays all relevant variable states.
- **Final Output:** Returns a debug-friendly horizontal array.

3 Why HAD is Useful

Key Points Behind HAD Debugging

- **Separation of Concerns:** Keeps core logic clean; adds debug as a separate section.
- **Principle of Least Knowledge:** Core logic does not need to know how debugging is handled.
- **Error Handling:** `echeck` turns errors into readable messages (e.g., "ERROR #VALUE!").
- **Visibility of Intermediate Values:** Shows all variables and intermediate results using `HSTACK`.
- **Standardized Boilerplate:** Ensures consistency and reduces setup time.
- **Non-Intrusive:** Debugging logic does not alter computation behavior.

Why Translate Error Values into Text?

- **Readability:** Text like "ERROR #VALUE!" is clearer than raw Excel error codes.
- **Prevents Error Propagation:** Errors won't crash the output array.
- **Efficiency:** You can immediately spot where things go wrong.
- **Consistency:** Outputs are easier to scan and test.
- **Compatibility:** Works with functions like `HSTACK` that don't handle errors well.

4 Theoretical Foundations: PLK and LoD

Principle of Least Knowledge (PLK)

- Components should only interact with a minimal number of other components.
- Promotes modularity, testability, and easier maintenance.

Law of Demeter (LoD)

- A concrete rule in OOP: "Only talk to your immediate friends."
- A method should only call:
  - Its own methods
  - Methods of passed parameters
  - Methods of objects it directly holds or creates

Connection

- **LoD is a specific instance of PLK.**
- Both aim to reduce coupling and make code more maintainable.
- Encourage encapsulation and minimal dependencies.

Origins

- **PLK:** A general design principle in software engineering.
- **LoD:** Introduced in the 1980s by Ian Holland and Karl Lieberherr at Northeastern University.

Why These Principles Matter

- **Reduced Coupling:** Modules can evolve independently.
- **Improved Maintainability:** Local changes don't ripple across the system.
- **Easier Testing and Debugging:** Smaller, well-scoped units are easier to test.
- **Scalability:** Systems with loose coupling are easier to grow.

Summary

- HAD debugging makes Excel `LAMBDA` functions more debuggable and maintainable.
- Error values are translated into readable text to support easier testing and reduce propagation.
- HAD follows the Principle of Least Knowledge, aligning with best practices in software design.
- These ideas build on foundational principles like the Law of Demeter and modular debugging techniques.

Additional Resource

Live Example: A working HAD-instrumented formula built through interaction with ChatGPT can be viewed at:

`https://chatgpt.com/share/a5ac0fce-2c5b-4281-8485-8664aaa48484`

This example demonstrates the full process of instrumenting an Excel `LAMBDA` function using `HAD`, following the principles outlined in this document.