Advanced Editor Tooling for Chapel

Daniel Fedorin and Jade Abraham

April 16, 2024
Advanced Editor Tooling

- Modern languages provide a variety of text editor tooling to writing code easier for the user
  - Syntax Highlighting
  - Code Intelligence
    - Hover information
    - Go-to-definition
    - Autocompletion
    - Type hints
  - Linting
  - IDE debugger support
  - Auto-formatters
Advanced Editor Tooling

- Modern languages provide a variety of text editor tooling to writing code easier for the user
  - **Syntax Highlighting**
    - Code Intelligence
      - Hover information
      - Go-to-definition
      - Autocompletion
      - Type hints
  - Linting
  - IDE debugger support
  - Auto-formatters

Until recently, Chapel has not provided such tooling.
Modern languages provide a variety of text editor tooling to writing code easier for the user:

- **Syntax Highlighting**
- **Code Intelligence**
  - Hover information
  - Go-to-definition
  - Autocompletion
  - Type hints
- **Linting**
- **IDE debugger support**
- **Auto-formatters**

With 2.0, Chapel provides new tools that help fill this gap!
Introducing....

**chpl-language-server**
- Provides Dyno-based code intelligence
  - Go-to-definition
  - View documentation
  - Inspect generics
  - ......

**chplcheck**
- Linter for common style issues and anti-patterns
- Used from the command line or from an editor
- Configurable with custom rules
- Automatically refactors code

```c
use List;
import BitOps.popCount;

config const lowBound: int(64) = -10;

show Generic | Show instantiation
proc numbers(low: int(64), high: low.ty)
  return low..high;
}

var bitLits: list(int(64), false) = new list(int);
for i: int(64) in numbers(lowBound, high = 10) {
  const numBits = popCount(i);
  bitLits.add(numBits);
}
```

```c
for (i, j) in zip(1..10, 1..10) {
  writeln("i = ", i);
  write("j = ", i);
  Lint: rule [IncorrectIndentation] violated
}

proc saxpy(n:int, s:int, a: [], b: []) {
  Lint: rule [UnusedFormal] violated
  var c: [0..#n] int;
  foreach i in [0..#n] {
    Lint: rule [SimpleDomainAsRange] violated
    Quick Fix
    Apply Fix for SimpleDomainAsRange
  }
}
```
How is this possible?

- Dyno!
  - Chapel’s compiler frontend rewrite
  - Among other things, provides:
    - Compiler-as-a-library (exposed to C++ and Python)
    - Incremental compilation

```
Program text

<table>
<thead>
<tr>
<th>Parsing</th>
<th>Type Checking</th>
<th>Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Executable</td>
</tr>
</tbody>
</table>

Abstract Syntax Tree

Type & Call Information

Python Bindings for Dyno

Dyno

chpl-language-server
chplcheck
Your tool here?
Dyno’s Python Bindings

- The Python bindings allow one to easily write language tooling that uses the compiler’s knowledge
- E.g., a function from ‘chplcheck’ written in terms of ‘chapel-py’ types

```python
@driver.basic_rule(NamedDecl)
def ChplPrefixReserved(context: Context, node: NamedDecl):
    """
    Warn for user-defined names that start with the 'chpl_' reserved prefix.
    """

    if node.name().startswith("chpl_"):
        path = node.location().path()
        return context.is_bundled_path(path)
    return True
```
Using LSP Tools

- Documentation for ‘chpl-language-server’:
- Documentation for ‘chplcheck’:
  - https://chapel-lang.org/docs/tools/chplcheck/chplcheck
- The Visual Studio Code extension:
Editor Tooling in Action!