Grammars in a broad sense
Grammars in a narrow sense

Which one?

- recognise programs in a language
- parse and interpret
- parse and translate
- parse and compile
- semiparse and analyse
- document
- domain model
- verify & validate
Grammars for IDE support

- idea from attending PLDI

- What is specific to grammars used in IDEs?
- What IDE features need grammar support?
- How to provide it better?

- OK to be “in a broad sense”
Main principles

- fast
- partial
- not limited by parsing in a narrow sense
Syntax highlighting

- colour-code tokens
- commonly implemented with regexes
- easy if the tokeniser is precise
- good luck with C++, PL/I, etc
- embryos of common interfaces
- novel solutions possible

```java
switch (state)
{
  case 1: // start of the file
    file = new CastleLanguage.Bind.BindFile();
    line = line.PadRight(51);
    if (!line.Begins("$$START "))
      {
        Error(1, line:lnr, pos:"1..8");
        state = 2;
        goto case 2;
      }
  var B1 = line.Substring(8,23);
  var str1_2 = line.Substring(31,14);
  if (str1_2!=" VER54 HPS541 ")
    {
      Error(1, str1_2, line:lnr, pos:"32..45");
      return null;
    }
  var E1 = line.Substring(45,6);
  if (!E1.In("AB30 ", "HPS531"))
    {
      Error(1, E1, line:lnr, pos:"46..51");
      return null;
    }
```
Code completion

- complete keywords
- suggest matching tokens
- guide indentation
- name suggestions
- drop down members
Word selection

- select a word, highlight “the same thing”
- “cheap” visualisation
- liked by devs
- not researched at all
Code folding

- blocks in composite statements
  - NOT a solution!
- hierarchical entities
- handful of top constructs?
Visual editing

- let graphs be graphs
- let tables be tables
- let window panels be window panels
- don’t let your dreams be dreams!
- projectional?
Debugging

- only relevant for executable programs
- step over
- step into
- breakpoint
- watch
- cross language boundaries
Testing

- discover tests
- running tests
  - live
- analysing tests
  - green and red
  - coverage
- incrementality
- why not advanced features?
  - model-based
  - test generation
  - fuzzing
  - mutants
Coding conventions

- formatting
- pretty-printing
- naming
- calling
- templates
- deprecating language constructs
- satisfying global constraints
- smell detection
Refactoring

- recommender systems
- ReSharer, CodeMaid, etc
- grammars are originally rewriting systems
- seldom used for rewriting
- can be insanely complex
- hard to do right
- hard to v&v
Navigating the codebase

- go to definition
- find references
- analyse dependencies
- analyse co-changes
Configuring a build

- compiling
- deploying
- delivering
- versioning
- building in the right order
Helping

- tooltips
- hover infoboxes
- API guidance
- explaining errors
  - recommending fixes

```csharp
file.Members.Add(block);

string x.Substring();
```

string string.Substring(int startIndex, int length)

Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.

**startIndex**: The zero-based starting character position of a substring in this instance.
Conclusion

- IDEs are built ad hoc
- IDEs are built with a framework bias
- there is [or can be] a class of IDE-specific grammars
- mostly greenfield research
- way beyond [single] grammars [in a narrow sense]
- vastly different user stories
  - JS: live to the extreme
  - C++: many changes, always incomplete info
  - C#: style and paradigm switching
  - PL/I: cache to the extreme (yesterday's trees are good enough)
- Please do it (willing to collab)