Structure/Structured/ Projectional Editors
Quick facts before today’s discussion

• CS performance is not bimodal
• Students who use projectional editors vs. textual editors perform equivalently on tests of CS knowledge
• For novices to a given language (not necessarily to CS!), projectional editors make users more productive

More on all of these in Thursday’s class, but we’ll have a more productive discussion today if we go over these before we start chatting :)
Reading Reflection

Discuss in groups

• What do you think is the difference between a visual editor and a projectional editor (if any)?
• Based on the videos:
  • What kinds of errors are projectional editors preventing?
  • What kinds of errors are projectional editors not preventing?
  • If you’ve ever used a projectional editor (or a projectional editing mode in a textual editor), what did you like or dislike about the experience?
structure editor

==

structured editor

==

projectional editor
What’s happening inside my compiler?

```
// square a couple numbers
int square(int num) {
    return num * num;
}
```

Parser

```
square  arg  int  return

num  int  ×

num  num
```

Code generator

```
square(int):

1     push    rbp
2     mov     rbp, rsp
3     mov     DWORD PTR [rbp-4], edi
4     mov     eax, DWORD PTR [rbp-4]
5     imul    eax, eax
6     pop     rbp
7     ret
```
What’s happening inside my compiler?

WHAT’S ALL THIS??

Parser

Compiler

fun

\[
\text{fun} \rightarrow \text{square} \rightarrow \text{arg} \rightarrow \text{int} \rightarrow \text{return} \\
\text{num} \rightarrow \text{int} \rightarrow \times \\
\text{num} \rightarrow \text{num}
\]

Assembly Language

1. `push rbp`
2. `mov rbp, rsp`
3. `mov DWORD PTR [rbp-4], edi`
4. `mov eax, DWORD PTR [rbp-4]`
5. `imul eax, eax`
6. `pop rbp`
7. `ret`

// square a couple numbers
int square(int num) {
    return num * num;
}
What’s happening inside my compiler?

Assembly Language

Compiler

AST

fun

square arg int return

num int ×

num num

parser

code generator

1 // square a couple numbers
2 int square(int num) {
3     return num * num;
4 }

square(int):
1 push rbp
2 mov rbp, rsp
3 mov DWORD PTR [rbp-4], edi
4 mov eax, DWORD PTR [rbp-4]
5 imul eax, eax
6 pop rbp
7 ret
Abstract because we’re not putting in every detail of the actual programming language syntax. (E.g., we’ve dropped all those pesky semicolons and parens.)

Syntax because we’re representing the syntactic structure of the code in question.

Tree because...well, obvious. But look, we got to throw away a bunch of parens and other grouping things because it’s all in the tree structure now!
Abstract Syntax Tree (AST)

Programs are data! We can mess with them!

…and we can build them up directly. We don’t have to write in a textual programming language and use a parser to recover this structure.
Projectional Editor

An editor where you’re building up the AST directly.

People can argue about the meaning of “directly.” How far does it have to be from the actual AST before it stops being a projectional editor? But basically it’s just a judgment call.
Projectional isn’t a feature of the programming language

It’s a feature of the programming environment!

Basically, it’s a matter of what editor we’re using to build up programs in the language.
```python
import weather
import matplotlib.pyplot as plt

celsius_temperatures = []
for t in weather.get_forecasts("Miami, FL"):  
celsius = (t - 32) / 1.8  
celsius_temperatures.append(celsius)
plt.title("Celsius Temperatures of Miami")
plt.plot(celsius_temperatures)
plt.show()
```
...also Python
Programming Language vs. Programming Environment

Both of those were Python—same language.

One editor was clearly textual, and one editor was clearly visual.

One editor was (probably) non-projectional, and one editor was clearly projectional.
Programming Language vs. Programming Environment

Programming Language: For our purposes today, a code generator that takes ASTs as input

Programming Environment: The tool or tools we use for building up those ASTs
Programming Language vs.
Programming Environment

Why do people get this confused?

Probably just because there are some languages that have only one interpreter, and the interpreter is embedded in a custom visual editor. If no one has written a parser for a text-based version of a given language, a visual environment may be the only way to write programs in it.
Programming Language vs. Programming Environment

Examples

Snap! : Both a programming language and a paired programming environment

Scratch : Same deal, both a programming language and a paired programming environment

Blockly : A library for making programming environments for whatever language you want
Projectional Editor vs. Visual Editor

**Projectional Editor:** Any editor (can be textual or visual) in which we build up programs by interacting directly with ASTs.

**Visual Editor:** Any editor (can be projectional or non-projectional) in which we build programs by any means other than typing text in a textbox.
Visual but not projectional

https://bubble.io/

build and run web applications without code
Visual but not projectional

Stagecast Creator™ allows adults and children as young as 8 to build their own simulations and games
Here are all the rules in Creator for "99 Bottles of Root Beer":

- Walk right
- Take one down...
- Walk left
- Start passing
- Drink
- ...and pass it around
- Dispose properly
syntax errors vs. logical errors
Before we switch to activity time...

Reminder to be working on final projects!!!
Snap! Activity

https://snap.berkeley.edu/snap/snap.html

Snap! Activity

We’re about to build some small extensions to a language that has a single interpreter that lives in a projectional editor.

Notes:

• This isn’t usually how people implement languages (even for languages intended to be used in projectional editors).

• However! It gives us a 2 birds 1 stone thing—we can experience using a projectional editor and building abstractions for use in a projectional editor at the same time!

• Intentionally slightly less directed than our usual activities, in hopes y’all will explore the Snap! landscape a bit.
Snap! Activity

• One tip before we get started.

• This seems like it shouldn’t matter, but it can get annoying, and no one ever figures it out themselves… If you end up with a “variable watcher” in the “stage” (white box in upper right) that you don’t want to show anymore, and you can’t get rid of it, drag it to the toolbox on the left that shows all the available blocks.
Snap! Activity - Stage 1

```
for each item in table
    add list bella 17 2 to db
    add list edward 104 1 to db
```
Snap! Activity - Stage 2

```plaintext
set db to (table
add list bella 17 2 to db
add list edward 104 1 to db
set column names of db to list name age other
set column names of db to list name age num_suitors
```

Make a block
table
add [ ] to
set column names of [ ] to [ ]
```
Snap! Activity - Stage 3

```python
set \( db \) to \( table \)

add list bella 17 2 to \( db \)

add list edward 104 1 to \( db \)

set column names of \( db \) to list name age other

set column names of \( db \) to list name age num suitors

set names to select column name from \( db \)

set ages to select column age from \( db \)
```
Reflection

• Was implementing database abstractions a difficult programming task? Hum for frustration level.
  • I’m so sorry. I know it’s wildly frustrating
• If you’re designing a new language, does putting it in a block-based editor automatically make it easy?
• Does getting out of syntax errors mean that you’re not programming?
• It’s probably been a while since you were a newcomer to programming, but this may have simulated some of that experience. If you’re designing for newcomers in your final project, or for people who are uncomfortable doing the particular kind of programming for which you’re designing, does this raise anything for you?
Snap! Activity - Stage 4

HW Assignment
Note: Doesn’t have to be in Snap!
(And don’t worry, just because we’re starting HW already
doesn’t mean we’re done with structure editors!)