Michael Winitch   Ben Snyder   Evan Tilley   Jason Cardinale
Sofía Sánchez-Zárate
the (ideal) team

Stephen A. Edwards

Stefén

S. Edwards

Stephen

TESTER

SYSTEM ARCHITECT

MANAGER

SYSTEM ARCHITECT

LANGUAGE GURU
the (actual) team

Michael
TESTER

Evan
SYSTEM ARCHITECT

Sofía
MANAGER

Ben
SYSTEM ARCHITECT

Jason
LANGUAGE GURU
origins

- There was once a guinea pig named bugsy...

He didn’t do too much, but everyone liked him and he is a good role model and our inspiration for ‘bugsy’, the language.

FUN FACT:
the ‘A’ in Stephen A. Edwards stands for AST!
outline

- “the team”
- bugsy overview
- compiler architecture
- testing
- classes
- arrays
- future work
- demonstration

FUN FACT:
the ‘A’ in AST stands for a**hole
JK -- amazing(;
bugsy overview

- a simple drawing language inspired by p5.js*
- object-oriented design using a blend of Python and Java syntax
  - classes, arrays, boolean logic
- allows for easy creation of shapes using an OpenGL backend
  - shapes: circles, ellipses, squares, rectangles, triangles, regular polygons, lines
  - animation: moveTo, rotateBy, scaleBy
  - stroke, stroke size, and fill: colors passed in as strings (ex: “0.3 0.6 0.1” RGB values)
- forget ints and floats – nums will ease your programming experience!
compiler architecture

SCANNER
Lexical Analysis

PARSER
Parsing

AST

Semantic Analysis
SAST

LLVM Generation
CODEGEN

SEMANT

BUILTINS

COMPILED CODE

COMPILER

LLC + GCC

LLVM

OPENGL LIB

OPENGL

Visual output
openGL library

- custom library connecting openGL to bugsy
- shape structures created to hold information about each type of shape
  - parameters: shape type, shape ID, x, y, r, w, h, x1, x2, y1, ...
- unique ID strings generated every time a new shape is created
  - used when animating, loops through array of shapes to check if we are redrawing the right shape at the right time/place
## OpenGL library

<table>
<thead>
<tr>
<th>OpenGL</th>
<th>bugsy</th>
</tr>
</thead>
<tbody>
<tr>
<td>display() (including glFlush() and everything in main())</td>
<td>draw()</td>
</tr>
<tr>
<td>glColor3f()</td>
<td>rgb()</td>
</tr>
<tr>
<td>glBegin(GL_QUADS)</td>
<td>rect()</td>
</tr>
<tr>
<td></td>
<td>square() with extra parameters</td>
</tr>
<tr>
<td>glBegin(GL_TRIANGLES)</td>
<td>triangle()</td>
</tr>
<tr>
<td>glBegin(GL_POLYGON)</td>
<td>regagon() with extra parameters</td>
</tr>
<tr>
<td>glutInitWindowSize() &amp;</td>
<td>canvas()</td>
</tr>
<tr>
<td>glutInitWindowPosition()</td>
<td></td>
</tr>
<tr>
<td>Custom function (we have one, another one</td>
<td>circle()</td>
</tr>
<tr>
<td>linked below)</td>
<td>ellipse() with different parameters</td>
</tr>
<tr>
<td>glBegin(GL_LINES)</td>
<td>line()</td>
</tr>
</tbody>
</table>
testing

- Test suite that compares an output to an existing file
- Challenge with testing visuals
- Approach: Add a print function to the OpenGL C code that prints out stats of the shape to confirm the program works as intended
- Pass in a DEBUG flag so that the window can close

```c
if(strcmp(getenv("DEBUG"), "1") != 0) {
    glutMainLoop();
}
```
nums

- Why num?
  - Simplicity and flexibility
  - Less need to worry about type errors

- Is this even possible?
  - Yes, thanks to build_fptoui
returning 0

- Successful main function should return 0 in LLVM
  - Always best to check in LLVM since that’s about as low as we are concerned for bugsy (one step above assembly code!)

C Program:    LLVM:

double main(){
    int x = 0;
    return 0.0;
}

define dso_local double @main() #0 {
    %1 = alloca i32, align 4
    store i32 0, i32* %1, align 4
    ret double 0.000000e+00
}

Don’t do this!
solution (pt. 1)

- Codegen!
  - Insert a return 0 at the end of the main() function:

```scala
let rec stmt builder = function
SBlock sl -> List.fold_left stmt builder sl
| SExpr e -> ignore(expr builder e); builder
| SReturn e -> ignore(match fdecl.styp with
  (« Special \"return nothing\" instr »)
  A.Void -> L.build_ret_void builder
  |
  («if a function returns an int (only main), build 0 return type »)
  A.Int -> L.build_ret (L.const_null i32_t) builder
  (« Build return statement »)
  | _ -> L.build_ret (expr builder e) builder );
```

```scala
57 type func_decl = {
58     mutable typ : typ;
59     fname : string;
60     formals : bind list;
61     locals : bind list;
62     fbody : stmt list;
63 }
```
solution (pt. 2)

- Does this work, and how do we know?
  - Yes -- LLVM!

```c
num main()
{
    num x;
    x = 5;
    return 0;
}
```

```cpp
define i32 @main() {
    entry:
        %x = alloca double, align 8
        store double 5.000000e+00, double* %x, align 8
        ret i32 0
}
```
arrays

Seems like it should be simple enough...

```plaintext
| SArrayAccess(a, e, l) -> let valu = (expr builder e) in
| L.build_load (L.build_gep (lookup a) [[L.const_int i32_t 0; valu]])
```

This won’t work… why?

Alright, seems like an easy enough fix… (cast as float)

```plaintext
| SArrayAccess(a, e, l) -> let valu = L.const_fptosi (expr builder e) i32_t in
| L.build_load (L.build_gep (lookup a) [[L.const_int i32_t 0; valu]])
```

Works fine for constant (i.e. arr[5])
arrays (pt.2)

- What about variables?
  - Difficult interfacing LLVM with moe

```c
let truncated = L.build_fptosi (valu) i32_t "aasf" builder in L.dump_value(truncated);
let result = L.build_in_bounds_gep (lookup a) [L.const_int i32_t 0; truncated] a builder in L.build_load result a builder;
```

```c
y = [1.7, 2, 4.3, 4, 5];
for (z = 0; z < 5; z++){
    print(y[z]);
}
```

QUOTE:

Hans Montero is *a ray* of sunshine

2 lines of code in 24 hours:

```
, i32 0, i32 2evan@plt-cs4115 ~/real/bugsy $ vim bug.bug
evan@plt-cs4115 ~/real/bugsy $ ./bugsy.native -c bug.bug 1> /dev/null
; ModuleID = 'Bugsy'
source_filename = "Bugsy"
  %x1 = load double, double* %x, align 8  %y2 = getelementptr inbounds [5 x double], [5 x double]* %y, i32 0, i32 fptosi (double %x1 to i32)Use of instruction is not an instruction!
  %x1 = load double, double* %x, align 8
LLVM ERROR: Broken module found, compilation aborted!
Aborted
```

Root of the problem: https://llvm.org/doxygen/Verifier_8cpp_source.html
future work

● group shapes → with classes!
● RGB color object rather than a string
  ○ rgb(100, 200, 40) vs. “0.5 0.2 0.1”
● irregular polygons
● simultaneous animations
  ○ combining rotation, translation, and scaling at once for one object
  ○ allowing multiple objects to be animated synchronously
● garbage collection
● inheritance
● exceptions

FUN FACT: 2 hours of sleep can be enough (or it was today anyway)!
future work: classes

- call-site adjustment
- method lifting constructors and class methods
- each instance has its own variables and can use the class methods

syntax ex:
```cpp
class ~bankAccount {
    string name;
    num bal;
    constructor(string n) {
        name = n;
        bal = 0;
    }
    void deposit(num amt) {
        bal += amt;
    }
}
```
```cpp
num main (){
    ~bankAccount b = new ~bankAccount(“Stephen”);
    b.deposit(5); // ~bankAccount_deposit(b, 5);
}
```

FUN FACT:
classes are more fun when they’re not over zoom!
lessons learned

- make sure the whole **pipeline** works before writing hundreds of lines of code on one file!! we ran into this when creating the library
- make more **progress** sooner → bugs come up and halt progress, we had an idealistic idea of how much work was left → cut features
- set **realistic** goals → we started with an idea to get a robotic arm to move, then thought we would try drawing chemical formulas, but it turns out drawing shapes was hard enough
- more **planning** in the early stages of the project
questions?

thank you for an amazing semester!!!