Venture
Start Yours

James Sands (js4597)
Naina Sahrawat (ns3001)
Zach Adler (zpa2001)
Ben Carlin (bc2620)
Introduction

“Adventure” - history of text-based adventure games

Venture - C-like imperative language, developed in Ocaml, compiles to LLVM

Provide developer tools to spend more time imagining, and less time coding.
Language Evolution & Process
Tools we used

1. GitHub
2. Google Docs
3. Facebook Messenger
4. Vim
5. OCaml
Team Roles and Project Timeline

James - System Architect
Zach - Language Guru
Naina - Project Manager
Ben - Tester

Feb 5, 2017 – May 9, 2017
Contributions to master, excluding merge commits
Language Architecture & Specifications
Language Syntax

C-like syntax

```c
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}

int main() {
    print(gcd(2, 14));
    print(gcd(3, 15));
    print(gcd(99, 121));
    return 0;
}
```
Program Structure and Scoping

Programs Structure:

- Globals
- UDFs
- Main

Scoping:

- Static

```c
int b;
int foo(){
    int a;
    a = b + 5;
    return a;
}

int bar(){
    int b;
    b = 2;
    return foo();
}

int main()
{
    b = 5;
    print(foo()); // prints 10
    print(bar()); // prints 10
    return 0;
}
```
Architecture Diagram
**Semantic Checker**

Semantically correct AST

Symbols table

```haskell
let rec expr = function
  | Int_Literal _ -> Int
  | BoolLit _ -> Bool
  | String_Lit _ -> String
  | Char_Literal _ -> Char
  | Id s -> type of identifier s
  | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
    (match op with
      | Add | Sub | Mult | Div when t1 = Int && t2 = Int -> Int
      | Equal | Neq when t1 = t2 -> Bool
      | LessThan | Leq | GreaterThen | Geq when t1 = Int && t2 = Int -> Bool
      | And | Or when t1 = Bool && t2 = Bool -> Bool
      | _ -> raise (Failure ("illegal binary operator " ^
        string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
        string_of_typ t2 ^ " in " ^ string_of_expr e)))
    )
  | Unop(op, e) as ex -> let t = expr e in
    (match op with
      | Neg when t = Int -> Int
      | Not when t = Bool -> Bool
      | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
        string_of_typ t ^ " in " ^ string_of_expr ex)))
    )
  | Noexpr -> Void
  | Assign(var, e) as ex -> let lt = type of identifier var
    and rt = expr e in
    check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
      " = " ^ string_of_typ rt ^ " in " ^
      string_of_expr ex)))
```

Testing

Success Case

Fail Cases

Automation
4115: THE GAME!

(our demo)

Prepare your brain for the most riveting ‘venture of all. This course....