Propeller
A Property Manipulation Language
Final Report

Isra Ali
"Tester"
isra.hamid_ali@tufts.edu

Gwendolyn Edgar
"Manager"
gwendolyn.edgar@tufts.edu

Randy Price
"Language Guru"
edward.price@tufts.edu

Chris Xiong
"System Architect"
lxiong01@tufts.edu

May 7, 2022

1 Introduction

Propeller is a language designed to write programs using the reactive programming paradigm. Programs written in Propeller usually act upon external events such as user inputs and environmental changes. Most prominently, it features a binding system: functions can be bound to properties (akin to members or fields in other languages) of objects (akin to records or structs in other languages) such that a change in the value of a property results in a call to each of its bound functions.

Propeller has multiple runtime environments: a simple runtime with a fixed entry point, and one that monitors certain local parameters of the computer (e.g. CPU temperature sensor readings). Given other runtime environments, Propeller can be extended to interface with any external input, such as that from a GUI, or from another kind of sensor.

2 Quick Tutorial

For those who have programmed in C-like languages, Propeller will feel familiar in an instant. Here's an example of a minimal Propeller program:

```
fn init() -> int
{
  prints('Hello World!\n');
  return 0;
}
```

Like C, Propeller uses curly braces to group statements. However, unlike C, these braces are mandatory even if the block only consists of only one statement.

```
fn foo(int n) -> int
{
  if n % 3 == 0
  { return 9; }
  else
  { return 42; }
}
```

Propeller doesn't have scoped local variables – all locals must be declared at the beginning of a function.

```
fn foo() -> void
{
  int i;
  i = 0;
  # int invalid_a;     invalid because decalaration appeared after a statement
  while i < 10
  {
    # int invalid_b;   invalid because this is block is not the beginning of a function
    print(i);
    i = i + 1;
  }
}
```

Defining a custom object is much like defining a struct in C:

```
objdef Jumbo
{
  str name;
  int age;
  float gpa;
}
```

Similarly, accessing and assigning to properties of objects is just like accessing and assigning to fields of a struct in C:

```
Jumbo jim;

jim.name = 'Jim';
jim.age  = 24;
jim.gpa  = 3.73;
```

Functions can be bound to properties, such that these functions are called whenever a new value is assigned to the property. Binding functions must take two arguments whose types match the type of the property.

```
fn celebrate(int old, int new) -> void
{
  print(new);
}

...

  bind(jim.age, celebrate);
```

3 Language Manual

3.0.1 Notion used in this Manual

This document uses a variation of Backus-Naur form to describe the syntax of the language. The most significant additions are the character range notation (`"a"-"d"` rather than `"a"|"b"|"c"|"d"`), and the notation for items repeated zero or more times (`{<characters>}` means `<characters>` repeated zero or more times).

3.0.2 Significant changes compared to the tentative version

- Escape sequences are now well-defined. Single quotes cannot be escaped.
- Strings are no longer syntactic sugar for int lists.
- Semantics of bindings have changed. For non-`external` objects, bindings are managed at compile time.
- Function binding only works for local object variables. Functions bound to internal objects receive two copies of the new value as their arguments.

3.1 Lexical Conventions

3.1.1 Comments

Propeller supports single-line comments. Any sequence of characters following a hash (`#`) that is not part of a string literal will be treated as a comment. Comments automatically terminate at the end of a line.

```
# This is a comment.

# valid identifiers
x
ready?
aBc123
o_o
AsDfG_1234_5?

# invalid identifiers
8eight
two__underscores
propeller_
too?early
huh_?
```

3.1.2 Identifiers

```
   <letter> ::= "A"-"Z" | "a"-"z"
   <digit>  ::= "0"-"9"
 <alphnum>  ::= <letter> | <digit>
```

```
<identifier> ::= <letter> ("?" | "")
                 | <letter> ("_" | "") {<alphnum> | <alphnum> "_"}
                   <alphnum> ("?" | "")
```

An identifier in Propeller is a character sequence consisting of letters, digits, underscores, and one optional question mark. Identifiers must begin with letters, cannot contain consecutive underscores, and cannot end with an underscore. Additionally, identifiers may end with a single question mark, but may not end with an underscore followed by a question mark.

3.1.3 Keywords

Propeller has 24 reserved keywords:

```
and     bind     break  continue elif
else    external float  fn       for
from    if       int    list     not
objdef  or       return str      to
unbind  void     while  xor
```

3.1.4 Separators

Propeller has 10 separators used to construct literals, define functions, separate statements, and more:

```
( ) [ ] { }
, . ;   ->
```

3.1.5 Operators

Propeller has 15 operators for comparison, logic, and arithmetic.

```
# comparison  logic  arithmetic
  =           and    +
  !=          or     -
  >           not    *
  <           xor    /
  >=                 %
  <=
```

3.1.6 Literals

```
      <int-literal> ::= {<digit>}
    <float-literal> ::= {<digit>} "." {<digit>}
```

```
     <bool-literal> ::= "true" | "false"
<string-characters> ::= character or escape sequence
   <string-literal> ::= "'" {<string-characters>} "'"
```

Supported escape sequences include `\n` for linefeed, `\r` for carriage return, `\t` for horizontal tabulation.

```
# int literals      float literals      boolean literals        string literals
1                   3.14                true                    'hello'
23                  0.78                false                   'h0wdy!'
0                   12.345                                      'Hello World!\n'
5839407430          0.999
```

3.2 Syntax

3.2.1 Expressions

```
<binary-operators> ::= "+" | "-" | "*" | "/" | "%"
                     | "==" | "!=" | "<" | "<=" | ">" | ">="
                     | "and" | "xor" | "or"
 <unary-operators> ::= "-" | "not"
       <expr-list> ::= <expression> | <expr-list> "," <expression>
        <arg-list> ::= "" | <expr-list>
      <expression> ::= <int-literal>
                     | <float-literal>
                     | <bool-literal>
                     | <string-literal>
                     | <list>
                     | <identifier>
                     | <expression> <binary-operators> <expression>
                     | <unary-operators> <expression>
                     | <identifier> "=" <expression>
                     | <identifier> "[" <expression> "]"
                     | "(" <expression> ")"
                     | <identifier> "(" <arg-list> ")"
                     | <identifier> "." <identifier>
                     | <identifier> "." <identifier> = <expression>
```

Operator precedence follows standard conventions (e.g. C-style precedence).

```
# the following are all syntactically correct expressions
1
abc
abc + def
alist[6]
zzz = true
(1 + 2) * 3
o.k = 'ok'
not false
zebra % horse
[true, false]
```

3.2.2 List Literals

```
<list> ::= "[" arg-list "]"
```

```
# int list
[1, 2, 3, 4]

# float list
[1.2, 3.4, 5.6]

# bool list
[true, false, true]

# str list
['joyce', 'cummings', 'center']

# empty list
[]
```

3.2.3 Declarations

Variable Declaration

```
              <types> ::= "int" | "float" | "bool" | "str" | "void" |
                          <identifier> | <types> "list"
      <variable-decl> ::= <types> <identifier> ";"
 <variable-decl-list> ::= "" | <variable-decl-list> <variable-decl>
```

Function Declaration

```
     <formal-list> ::= <types> <identifier>
                     | <formal-list> "," <types> <identifier>
 <formal-list-opt> ::= "" | <formal-list>
   <function-decl> ::= "fn" <identifier> "(" <formal-list-opt> ")" "->"
                       <types> "{" <variable-decl-list>
                       <statement-list> "}"
```

Object Type Declaration

```
          <obj-def> ::= "objdef" <identifier> "{" <variable-decl-list> "}"
 <external-obj-def> ::= "external" <obj-def>
```

```
# defining an object type "Patient"
objdef Patient
{
  str   name;
  int   age;
  float height;
  float weight;
}
# an external object type
external objdef ExternObj
{
  str name;
  int value;
}
# declaring a variable of type Patient
Patient p;

# declaring variables of other types
int x;
str name;
float pi;
bool is_ready?;
float list color;

# a minimal function that does nothing
fn do_nothing() -> void
{
  return;
}
```

3.2.4 Statements

Sequencing

```
<statement-list>  ::= "" | <statement-list> <statement>
<statememt-block> ::= "{" <statement-list> "}"
```

Control Flow

- Branching

```
      <if-statement> ::= "if" <expression> <statement-block>
                         <elif-statements> <else-clause>
   <elif-statements> ::= {"elif" <expression> <statement-block>}
       <else-clause> ::= "" | "else" <statement-block>
```

  Else clauses are attached to the closest unmatched if clause before it.

- Loops

```
    <for-statement> ::= "for" <identifier> "from" <expression>
                        "to" <expression> <statement-block>
  <while-statement> ::= "while" <expression> <statement-block>
```

- Jumps

```
    <break-statement> ::= "break" ";"
 <continue-statement> ::= "continue" ";"
   <return-statement> ::= "return" ("" | <expression>) ";"
```

Special Statements Built-in functions `bind` and `unbind` have special syntactical rules and semantics, but can be viewed by the user as regular functions.

```
   <bind-statement> ::= "bind" "(" <identifier> "." <identifier> ","
                        <identifier> ")" ";"
 <unbind-statement> ::= "unbind" "(" <identifier> "." <identifier> ","
                        <identifier> ")" ";"
```

All Statements

```
 <statement> ::= <expression> ";"
               | <if-statement>
               | <for-statement>
               | <while-statement>
               | <break-statement>
               | <continue-statement>
               | <return-statement>
               | <bind-statement>
               | <unbind-statement>
```

```
# example for statements
if x > 0
{
  prints('Positive');
}
elif x == 0
{
  prints('Zero');
}
else
{
  prints('Negative');
}

while x < 10
{
  prints('Less than 10')
  x = x + 1;
}

sum = 0;
for ii from 1 to 1000000
{
  sum = sum + ii;
  if sum < 0
  {
    break;
  }
}
```

3.3 Semantics

Propeller's operational semantics is heavily inspired by C-like languages. For the sake of brevity, commonplace semantics found in other widely-used languages are omitted in favor of the semantics guiding Propeller's most prominent features.

3.3.1 Data Types

Primitive Types Primitive types use the following internal representations:

Type	Size (bytes)	Description
`int`	4	Integer
`float`	8	IEEE 754 floating point
`bool`	1	Boolean
`str`	varies	Stores UTF-8 encoded characters of a string
`void`	N/A	Only used as return type for functions that return nothing

Lists `list`s contain one or more elements of the same type; the elements are immutable, and can be indexed with separators [ ]. Elements of a list are stored sequentially in memory. For example, a `bool list` of length 5 takes up 5 bytes in memory, with the first element stored in the first byte, the second element in the second byte, and so on. Attempting to index a list outside of its bounds results in undefined behavior. Although the elements of a list are immutable, a list variable may be overwritten by assigning another list to it.

Objects Propeller allows users to define custom types called objects. An object has one or more properties, which are variables of primitive types. Additionally, functions may be bound to these properties and executed upon a change in the property's value. A runtime can have several predefined objects from an external library, but these objects must be defined and prefixed with the `external` keyword. Object variables may not be passed to functions.

3.3.2 Operations

`int` and `float` Operations Comparison and arithmetic operators are overloaded in Propeller. Comparing an `int` to a `float` will result in the promotion of the `int` expression to a `float` expression; similarly, combining an `int` and a `float` with a binary arithmetic operator will result in the promotion of the `int` expression to a `float` expression, and the result of the arithmetic operation will be a `float`. Additionally, the modulus `%` operator, only takes `int` operands, and the divison of an `int` by a `int` returns a `float`.

The modulo operator follows the "truncate" definition.

```
# comparison
2.3 == 1;   # false
4 != 5;     # true
8.34 > 3;   # true
9 < 10;     # true
5.5 >= 5.6; # false
100 <= 100; # true

# arithmetic
4 + 3;    # 7
2.0 - 1;  # 1.0
-3.3 * 3; # -9.9
5 / 2;    # 2
5.0 / 2;  # 2.5
6 % 4;    # 2

# boolean comparison
true != false; # true
false == true; # false

# boolean operations
not false;      # true
true and false; # false
true or false;  # true
true xor true;  # false
```

Lists can be indexed using the `[]` operator.

```
# list indexing
int list l = [1, 2, 3];
l[1];           # 2
```

3.3.3 Control Flow

Propeller uses `if/elif/else` clauses, `for` loops, and `while` loops. Each control flow method in its entirety is a statement, and the body of an `if/elif/else` clause or loop is comprised of a list of statements. Control flow semantics are C-like, with the following differences:

- `if/elif/while` expressions need not be enclosed in parentheses.
- Statements following `if/elif/else`, `for`, and `while` must be enclosed in curly braces { }.
- There is no such thing as a "block" - each control flow method is followed by a list of one or more statements enclosed in curly braces.
- `for` loops have a unique syntax, and are intended to execute a given number of times. When writing a `for` loop, the name of the looping variable must be given, along with integer expressions that evaluate to the looping variable's initial and final value, respectively. When the looping variable is greater than the final value, the loop terminates.

- Keywords `break` and `continue` can be used to jump out of a loop or continue to its next iteration, respectively.

3.3.4 Binding

Functions can be bound to properties of objects such that whenever the property is assigned a value, the functions bound to that property are called.

Let β be the bindings that are currently established during execution of the program. β is one of the environment metavariable of Propeller's operational semantics. $\beta(o, p)$ is a set of functions bound to property p of object o. Note that this way objects of the same custom type don't share bindings.

A function bound to a property must accept two parameters: two values of the same type as the property itself, passing the old value of the property and new value of the property respectively.

When multiple functions are bound to the same property of an object, their order of execution is defined by the order in which they were bound.

Semi-formal operational semantics of syntactical forms related to binding will be given below. $\rho(o, p)$ retrieves the location where property p of object o is stored, and $\sigma(l)$ is the value at location l.

$$\frac{\begin{array}{c}\langle e, \rho, \sigma, \beta, \cdots\rangle \Downarrow \langle v, \rho, \sigma_0, \beta, \cdots\rangle \\ \text{for each } f_i \in \beta(o,p), i = 1 \ldots n \\ \langle f_i(\sigma_0(\rho(o,p)), v), \rho, \sigma_{i-1}, \beta, \cdots\rangle \Downarrow \langle \mathit{void}, \rho, \sigma_i, \beta, \cdots\rangle\end{array}}{\langle \textsc{PropertyAssign}(o, p, e), \rho, \sigma, \beta, \cdots\rangle \Downarrow \langle v, \rho, \sigma_n\{\rho(o,p) \mapsto v\}, \beta, \cdots\rangle} \quad \textsc{PropertyAssign}$$

$$\frac{}{\langle \textsc{bind}(o, p, f), \rho, \sigma, \beta, \cdots\rangle \Downarrow \langle \mathit{void}, \rho, \sigma, \beta\{(o,p) \mapsto \beta(o,p) \cup \{f\}\}, \cdots\rangle} \quad \textsc{Bind}$$

$$\frac{}{\langle \textsc{unbind}(o, p, f), \rho, \sigma, \beta, \cdots\rangle \Downarrow \langle \mathit{void}, \rho, \sigma, \beta\{(o,p) \mapsto \beta(o,p) \setminus \{f\}\}, \cdots\rangle} \quad \textsc{Unbind}$$

The semantics above only applies if the object in PropertyAssign is defined as `external`. For non-`external` objects, bindings are managed statically at compile time – they are effective for all statements that are parsed in between the `bind` statement and its corresponding `unbind` statement.

Due to limitations of the implementation, bindings only work for local variables. In addition, functions bound to non-`external` objects do not recieve the property's previous value when called. Finally, external objects cannot have values assigned to their properties.

3.3.5 Program Execution

When a program written in Propeller is executed, it begins from a function called `init()`. After `init()` returns, it enters an event loop defined by the runtime library. For the most basic text-mode only runtime, the event loop simply terminates the program.

3.4 Built-in Functions and Runtimes

Propeller has a minimal set of built-in functions for printing to the standard output:

`print(int a)`	Prints an integer value to the console and starts a new line
`printb(bool a)`	Prints a boolean value to the console and starts a new line
`printf(float a)`	Prints a float value to the console and starts a new line
`prints(str a)`	Prints a string to the console

Propeller comes with two different runtimes: A default one that just runs the entry function and does nothing interesting, and one that monitors the ACPI thermal zone of a computer.

The sensor library provides the following object definition:

```
1 external objdef Sensor
2 {
3   int temperature;
4 }
```

The value stored in `temperature` is the value read from the temperature sensor in 1/1000 of a degree Celsius. Because a Sensor is an external object, its temperature field cannot be programatically assigned a value. Instead, its value is updated every second in the runtime environment. Due to the limitation of the implementation, only one function can be bound to this property.

The sensor runtime only works on Linux with ACPI thermal zone support (the file `/sys/class/thermal/thermal_zone0/temp` must exist on the system).

3.5 Examples

3.5.1 Minimal test driving program

This is a basic text-mode program. It should print "25," then terminate. It uses the basic text-mode runtime environment.

```
objdef Jumbo
{
  str name;
  int age;
  float gpa;
}

fn celebrate(int old, int new) -> void
{
  print(new);
}

fn init () -> int
{
  Jumbo jim;

  jim.name = 'Jim';
  jim.age  = 24;
  jim.gpa  = 3.73;

  bind(jim.age, celebrate);

  jim.age = 25;

  unbind(jim.age, celebrate);

  jim.age = 26;

  return 0;
}
```

3.5.2 Temperature Monitor

This program models a simple temperature monitor that prints a message when the temperature reading exceeds a threshold. This program will require the correct runtime environment (`sensor_linux`).

```
external objdef Sensor
{
  int temperature;
}

fn print_warning(int oldt, int t) -> void
{
  if (t > 60000) and (t != oldt)
  {
    prints('thermal zone sensor readout too high: ');
    print(t / 1000);
  }
}

fn init() -> int
{
  Sensor sensor;
  bind(sensor.temperature, print_warning);
  return 0;
}
```

4 Project Plan

4.1 Process used

Our team used git for source code version control. In general, team members worked on their tasks individually. All team members have basic knowledge of git, and had direct access to the main repository (`https://github.com/gfaline/Compilers`).

The team held irregular meetings to coordinate strategies for completing each deliverable. Initially, we used GitHub issues to delegate and assign tasks, but as the semester progressed, we moved most activity to a Discord server. Our advisor, Mert, has access to a special channel in this server, where we fielded several questions about design and implementation throughout the semester.

Specification documents can also be found in the git repository. Early drafts were created using the LaTeX collaboration platform Overleaf.

4.2 Style guide

As the members worked on their own, there is no explicit style guideline. The untold rule is to follow the style of existing code. Most observed rules are:

- Indent with two spaces.
- Each line should not exceed 100 characters (not strictly enforced).
- Name variables descriptively, unless it's only used by a few expressions that follow it.
- Test as soon as the code is working.
- Put the "in" of a let statement at the end of the line if it's a variable, and on a new line if it's a function

4.3 Project timeline

Weekly commit history (generated with `git-bars`):

```
281 commits over 10 week(s)
2022/18  39   **********************
2022/17  1
2022/16  59   ***********************************
2022/15  11   ******
2022/14  6    ***
2022/12  33   ******************
2022/09  12   *******
2022/08  85   **************************************************
2022/07  34   ********************
2022/06  1
```

The project is very clearly deadline-driven. Regular project check-ins helped push the project forward, as evidenced by the spike in activity around each one.

The task list that laid out by the team at the beginning of the semester is listed below:

```
[ ] Hello World (print an integer)
 [ ]   SAST for expressions with only integer literals
 [ ]   dummy function call semantics & SAST (no type checking)
 [ ]   dummy semantics for return
 [ ]   stubs for statement sequencing and statement blocks (without typing etc)
 [ ]   code generation for function call and return
 [ ]   built-in function print_int
 [ ]   script for linkage and other shenanigans (generating the executable)

[ ] SAST for expressions
 [ ]   arithmetic operations between integers
 [ ]   arithmetic operations between floats
 [ ]   arithmetic operations between floats and integers (promotion)
 [ ]   comparison between integers and floats (promotion, can be split like above)
 [ ]   boolean operations
 [ ]   boolean comparison
 [ ]   type checking for all invalid cases

[ ] Expression & code generation (without variables)
 [ ]   code generation for integral arithmetic operators
 [ ]   code generation for float operators
 [ ]   code generation for boolean operators
 [ ]   code generation for comparison operators
 [ ]   assignment to variables: code generation
 [ ]   built-in function print_bool and print_float
 [ ]   tests & test scripts

[ ] Variables
 [ ]   declaration and allocation
 [ ]   typing variable access expressions
 [ ]   typing assignment to variable expressions
 [ ]   code generation
 [ ]   tests

[ ] If statements
 [ ]   type checking & SAST
 [ ]   code generation
 [ ]   tests

[ ] For loops
 [ ]   formal semantics
 [ ]   type checking & SAST
 [ ]   code generation
 [ ]   tests
```

```
[ ] While loops
 [ ]   type checking & SAST
 [ ]   code generation
 [ ]   tests

[ ] Jumps
 [ ]   break semantics & code generation
 [ ]   continue semantics & code generation
 [ ]   return semantics & type checking
 [ ]   return SAST & code generation
 [ ]   tests

[ ] Functions
 [ ]   arguments type checking
 [ ]   return value type checking
 [ ]   SAST
 [ ]   code generation
 [ ]   tests


**Tentative below**
(Tasks with an asterisk are deemed "optional")


[ ] Lists
 [ ]   typing expressions with lists
 [ ]   internal representation, declaration and allocation
 [ ]   built-in functions for lists (part 1, empty?, add)
 [ ]   built-in functions for lists (part 2, first, rest)
 [ ]   strings
 [ ]   built-in functions for strings (print_str)
[ ]   tests

[ ] Objects
 [ ]   declaration & in-memory presentation
 [ ]   assignment to objects (value assignment only, no calling of bound functions)
 [ ]   accessing properties
 [ ]   objects as formal arguments (all pass by reference)
[ ]   tests

[ ] Bindings
 [ ]   extra internal structures for storing the list of bound functions
[ ]   code generation for bind
[ ]   code generation for unbind
[ ]   additional code generation for assignment to property of objects
```

```
[ ]*  cycle binding detection
[ ]   tests

[ ]*External objects
[ ]*  alternative code generation for assignment to property of external objects
[ ]   tests

[ ] Runtime environment
[ ]   other built-in functions (math etc)
[ ]   standard libraries
[ ]   basic text-mode runtime
[ ]*  qt based GUI runtime
[ ]*  runtime for the sensor example
[ ]   testing example code & additional tests
```

Tasks were split among the team members according to this list, but were not strictly enforced or adhered to.

4.4 Member roles

As shown in the authors list, we assigned a role to each member of the team:

- Tester: Isra Ali
- Manager: Gwendolyn Edgar
- Language Guru: Randy Price
- System Architect: Chris Xiong

The roles assigned came from the recommendation of the instructor and were assigned according to each member's preference. However, these assignments ended up being rather arbitrary - team members ended up working on their own modules individually, and became responsible for the modules' design and implementation. Important design choices were discussed in the Discord server before any final decisions were made.

4.5 Development environment

The project is developed and tested on:

- Gentoo Linux amd64, OCaml 4.14.0, LLVM 14.0.1 with OCaml binding installed with system package manager
- Ubuntu 20.04 LTS on Windows Subsystem for Linux, Windows 10, OCaml 4.13.1, opam 2.1.0, LLVM 10.0.0
- MacOS Monterey, M1 chip, ocaml 4.13.1, opam 2.1.2, llvm 13.0.1
- Kali GNU/Linux 2021.4, 5.10.16.3-microsoft-standard-WSL2 , Windows 10, OCaml 4.13.1, opam 2.1.2, LLVM 13.01

Git is the source code version control used. All documentation is typeset with LaTeX.

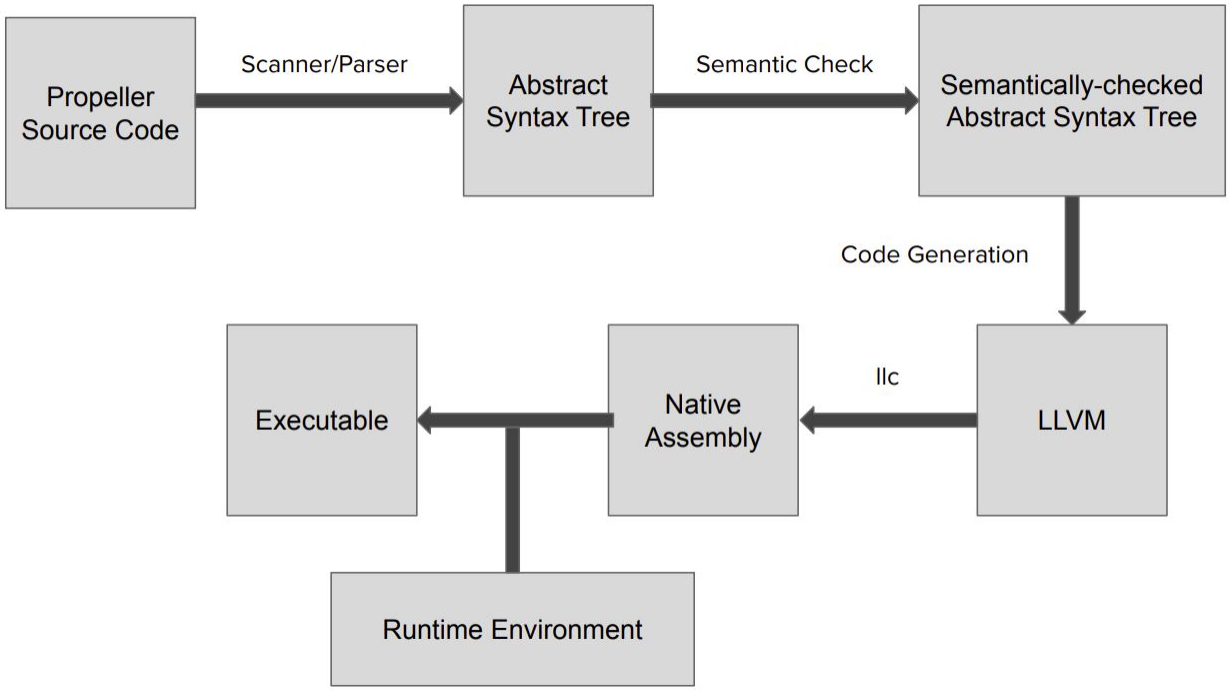
4.6 Project log

See Appendix A for the full git log.

5 Architectural Design

5.1 Compiler

The compiler follows the structure outlined in the lectures, with the additional steps that links the resulting object code against a runtime to generate the final executable.



5.2 Scanner

The scanner transforms a Propeller program into a series of tokens, which are separated by whitespace in the source code. Examples of tokens include variable names, function names, types, separators, operators, control flow statements, and literals. Additionally, the scanner throws a unique error if it encounters an ill-formed identifier (e.g. `h?3__llo`).

5.3 Parser

The parser receives a series of tokens from the scanner, then uses these tokens to construct an abstract syntax tree. Nothing of particular interest happens during the generation of a Propeller program's AST.

5.4 Semantic Checker

The semantic checker receives the AST from the parser, then performs various checks on declarations, definitions, expressions, and statements. It prevents the programmer from defining two or more functions with the same identifier, defining two or more variables in the same scope with the

same identifier, defining two or more object types with the same identifier, overwriting a built-in function's definition, using undefined identifiers, using undefined object properties, attempting to access a property of a non-object variable, binding a function to a property whose type does not match those of the function's formal parameters, binding a function with some `n != 2` formal parameters to an object's property, binding/unbinding a function to non-object variables, declaring variables of type `void`, creating list literals of a non-primitive type, creating list literals whose elements are of different types, passing an incorrect number of arguments to a function, passing one or more arguments to a function whose types does not match those of the function's formal parameters, returning a value whose type does not match the function's return type, assigning a value to a variable whose type differs from the value's type, and using unary and binary operators on expressions with inappropriate types, and using `continue` or `break` statements outside of loops. Once these checks are performed, the semantic checker generates a semantically-checked abstract syntax tree.

5.5 Code Generator

The code generator receives the SAST from the semantic checker, then converts the SAST into an LLVM IR of the original Propeller program. Several additional checks are performed as the instructions are being generated, which prevent duplicate bindings of a function to the same property, prevent the unbinding of a function from a property if it is not already bound, and assignment of properties to external objects.

5.5.1 Bindings

Bindings are processed in the code generation phase. It keeps track of which functions are bound to the properties of object variables using a single string map, which maps a combination of object variable names and their properties to a list of function names.

5.6 External objects and Runtime Environment

External objects are treated uniquely by Propeller. They are not objects that occupy memory spaces in the Propeller module; instead, all Propeller code treats them as integer values, like an identification for the corresponding instance.

The sole purpose of runtime environments in Propeller is to provide implementation of external objects. All interactions with external objects within propeller are translated into calls to functions in the runtime environment.

For each external object type, the runtime environment must implement
`int object_new_<typename>()`, which creates an object of `typename` and returns its identification.

For each property of object type `typename`, four functions must be implemented:

- `prop_t object_prop_get_<typename>_<propname>(int id)`: Gets the value of property `propname` of object with id `id`. The return type must match that of the property.
- `void object_prop_assign_<typename>_<propname>(int id, prop_t val)`: Assign `val` to the value of property `propname` of object with id `id`. This is not actually implemented because we don't have a use for it since GUI support is cancelled.

- `void object_prop_bind_<typename>_<propname>(int id, funcptr_t func)`: Bind a function to the property `propname` of object with id `id`. `func` is a function in the Propeller module and is guaranteed to take two parameters of the same type of the property.
- `void object_prop_unbind_<typename>_<propname>(int id, funcptr_t func)`: Same as above, but unbinds the function.

Since Propeller does not have garbage collection capabilities yet, the runtime cannot delete any of the objects it creates.

The runtime environment is required to run the entry point to the Propeller module `init()` on startup, then it can run any routine that's needed to notify the Propeller module of changes of property values. They are usually implemented as an "event loop", either with polling or wait for some system calls to return.

5.7 Component-level work split

- Lexer: Randy
- Parser: Randy
- Semantic Analysis:
  - base SAST transformation: Randy
  - function: Randy
  - operators, expressions: Randy
  - lists: Isra and Randy
  - strings: Isra and Chris
  - if statement: Randy
  - while statement: Randy and Chris
  - break and continue statement: Chris
  - for statement: Randy
  - return statement: Randy and Chris
  - built-in functions: Randy and Isra
  - objects: Randy
  - external objects: Chris
  - bindings: Randy
- Code generation:
  - function: Randy
  - return: Chris
  - integer literals: Randy
  - boolean literals and operators: Gwendolyn
  - modulo operator: Chris

  - float literals: Randy
  - other binary and unary operators: Randy
  - if statement: Randy
  - while statement: Randy and Chris
  - break and continue statement: Chris
  - for statement: Randy
  - list literals: Isra
  - objects: Randy
  - strings: Isra
  - bindings: Randy
  - external object bindings: Chris
  - built-in functions: Randy, Isra and Chris
- Runtime: Chris

6 Testing

All testing facility is located in the `tests` directory of the compiler source code.

6.1 Example source code and generated LLVM IR

See Appendix B.

6.2 Test scripts

See Appendix C. Each function is documented in the comment above it.

6.3 Test Automation

A 105-style testing interface (notably the name "CheckExpect") was used. The testing interface as well as all test scripts are written in Bash. Three different aspects of execution result can be checked against a given standard: standard output, standard error and return code. The first two can be ignored if needed. Since the compiler exits with non zero return code if an invalid program is fed to it, it can be used to verify test cases in which the compilation is expected to fail.

There are convenience wrapper functions of `CheckExpectWReturnCode` provided for different tasks to simplify the process to make a test suite.

For each test suite, a test script is created. Running these test scripts will run all the tests in that test suite.

6.4 Testing task split

The current testing framework is written by Chris. Parser unit tests are provided by Gwendolyn and Isra. All team members contributed to the extended test suite. Previously the project used a testing script written by Gwendolyn that was based on the test script of MicroC.

7 Lessons learned

7.1 Isra

- This class gave me a new perspective on programming and a deeper understanding of how the languages I use work. I've taken theory based classes like 170, but I didn't fully understand how what we learned about finite automata and context free grammars was applicable until now. It can be really useful to know how things work under the hood as a programmer.
- The scanner, parser, and semantic implementation wasn't very hard to understand, but code generation and llvm is much more complex. The documention for llvm isn't the best either. It has a steep learning curve, so it's best to start understanding codegen as soon as possible.
- Starting earlier would have been beneficial. I think we should have set strict deadlines about when we want the smaller features to be implemented throughout the semester, so we wouldn't have been as stressed before the larger deadlines.

7.2 Gwen

- Keep on track from the start. It is easy to get lost in the periods of time between assignments. It's also easy to get lost in how long things will take. I wish we were better about having smaller internal deadlines but life was very in the way for all of us.
- Functional programming forces me to work a little differently
- Compilers were a total foreign body to me. Going through first the labs them implementation helped me understand more of this black magic.
- Don't be afraid to cut what isn't working and isn't necessary. Scaling down and deciding what the most important bits are help to focus the work.

7.3 Randy

- Functional programming is super cool, but it has its limits. We resorted to using a StringMap reference (which behaves like a global variable of sorts) to keep track of what functions are bound to the properties of object variables. I'm sure there's a functional way to do that, but this approach made the most sense to us.
- Start early, on everything! Code generation can get particularly hairy.
- The OCaml LLVM API can be pretty confusing, so be sure to ask your advisor a bunch of questions if you can't figure out how to get something working.
- Temper your expectations from the beginning. Your language should only have a handful of core features are unique and super exciting - don't promise too much!

7.4 Chris

- Even after been warned multiple times, I still have the habit of not starting the real work until the last minute.
- Functional programming is fun in a way that it forces me to think in completely different patterns.
- (As advice for future students as well) Learn to manage the expectations. As initially planned, Propeller was clearly way too ambitious, even with GUIs, high-level list operations and all the bells and whistles. As time flew by, some of the goals are clearly unrealistic for a semester-long project and had to be dropped.
- (When implementing external object bindings) If searching something gives no useful results, and all hope is lost, trying all possible combinations could be a good way to go.
- Test-oriented development can be beneficial, but could result in a compiler that doesn't work with code that uses any untested features.
- Function pointers in LLVM causes even more confusion than function pointers in C++.
- I feel like as a team made of complete strangers, we could use more communication throughout the project.

8 Appendix A: Full Git Log

```
commit 3545e6da4ba101248f0b5eab3e230309db5335f2
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 21:49:48 2022 -0400

    Add final report.

commit 641256ceb066d25d952cba803350f2b88cd8abc2
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 20:06:40 2022 -0400

    Update readme.

commit 80b3437c49f4eba727a9ca29efaff6bee8da5dfd
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 20:03:59 2022 -0400

    Add author list to all OCaml source files.

commit fe849f5c2493275bad3a9bd18b715ca201f9e55d
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 19:38:07 2022 -0400

    Update readme and description for new tests.

commit e27ca020f25e027f3f7b229c36a6844202a19cd9
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 19:28:32 2022 -0400

    Add makefile and make targets for demo.

commit 857c7d8bc50aa8b9042a847150a7bf04fe3c761d
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 19:20:43 2022 -0400

    Move demo code to its own directory.

commit 2920e1656ea29d9d3d0446c2599b8bf7f0cf01bf
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 19:14:05 2022 -0400

    Fix all tests that doesn't work.

commit 714515ee6527d6ce47cf06fccfde6ef39bf35501
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 18:35:20 2022 -0400

    Tidy up the demo code a bit.

commit caaef3ba6ff55dbf0084c3b788c76ef629bf2f9c
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat May 7 16:50:20 2022 -0400

    Add more detail to some error messages.

commit 2fdc64ab9041633c586c79339dfc6653cd91fddc
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat May 7 16:50:01 2022 -0400

    Add presentation slides.

commit 0adcdcf6bf15153b0dcc99d655d08479caa8f7b7
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat May 7 13:50:24 2022 -0400
```

```
    Refactor binding-related stuff in codegen.ml

commit 87a2018218c6f72a066fe4d571c7e3fdbb9db03e
Merge: 9543666 981fce2
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Sat May 7 13:17:10 2022 -0400

    Merge branch 'main' of https://github.com/gfaline/Compilers

commit 954366669afebc0615fdbb3af5acd6516af920a0
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat May 7 13:17:04 2022 -0400

    Refactor SCall

commit bdb51b6c37ff410f3ad992a4a52bb2ec475ec330
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat May 7 13:07:47 2022 -0400

    List indexing works! Code cleanup to come.

commit 981fce2ae4a675615b5f9e4e36f01240e57a4d10
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 12:41:49 2022 -0400

    Implement all interfaces in the sensor runtime.

commit 3e4be31de3db20d3a00a271367096862497283fe
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 12:41:09 2022 -0400

    Update the sensor demo.

commit a43d90b632486556796f7ee20facb796b9bdfe42
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 12:39:04 2022 -0400

    Fix prop get for external objects.

    Mark assignment to external objects as unimplemented.

commit 0a1a4f91bddd1112169058c1dbb4db867553277a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat May 7 12:03:15 2022 -0400

    Remove redunant wildcard case in stmt.

commit 22bcc2c99c20056e4a117c06771e22608b5ce995
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 11:09:59 2022 -0400

    Strip away the quotes and unescape special characters in strings.

commit de9e239b488ce0994640050b578e63877c8a2f84
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sat May 7 11:09:03 2022 -0400

    Whoopsie, got the parameters backwards.

commit 666ce2c013243fef61e3ebc7d74944fafcab824e
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Sat May 7 06:17:02 2022 -0400

    Rename propeller/tests/test-list.exp to propeller/tests/exttest/test-list.exp
```

```
commit 20a547bb1c6da5d60d09512cda3ca23f1ace7f87
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Sat May 7 06:16:39 2022 -0400

    Rename propeller/tests/test-list.pr to propeller/tests/exttest/test-list.pr

commit 437a86dab9db7d1401a61d8465a40deea9747de4
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Sat May 7 06:16:01 2022 -0400

    Create test-list.exp

commit b28f1fa7326b944826372a4a27b2488462845fff
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Sat May 7 06:15:19 2022 -0400

    Create test-list.pr

commit 221ccc4591a55631ecd1f90a39f322428803c698
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Fri May 6 08:23:57 2022 -0400

    Create test-print.exp

commit 5a67ca20c5841ce77c7fb87cdfdf063340523137
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Fri May 6 08:23:24 2022 -0400

    Create test-print.pr

commit 85f59e9a1769f3eda32ff384af8ce24f27658350
Merge: c6f30eb aa9eaf7
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Fri May 6 08:21:50 2022 -0400

    Merge branch 'main' of https://github.com/gfaline/Compilers

commit aa9eaf7d5ca1a03fd63e71b600623e647ec283dd
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Fri May 6 08:21:40 2022 -0400

    Added print functions

commit c6f30eb91d44c7dd0501db5f0da4cdb3b4df5844
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Fri May 6 08:21:32 2022 -0400

    add silly demos

commit 78c13d76604df4254a2fc27fc0397bf9546462cf
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Fri May 6 08:19:24 2022 -0400

    List and print functions

commit cebe078a99a169c538cc144c6a07585c1cd78f24
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Fri May 6 02:27:53 2022 -0400

    Implement external objects. Fix LLVM error that occurs if a function has no return value.

    Added sensor demo.
    Added the runtime system.
    Renamed entry point (main -> init).
```

```
    Updated tests to reflect these changes.

commit dc841cddc99f0c79f85364a1de0f28be5913b12d
Merge: 107e6cc 73255a0
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Thu May 5 18:43:45 2022 -0400

    Merge branch 'main' of https://github.com/gfaline/Compilers

commit 107e6ccf9c824298f7596a74984e094e1a69b9bf
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu May 5 18:41:29 2022 -0400

    Add silly but kind of functional implementation of bind/unbind.

commit 73255a0000559cab54851f7cf20b66b32950c7dd
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu May 5 04:25:25 2022 -0400

    Updates SSliteral, scall print, and index

    Printing values other than integers causes an argtype error. Index is close to working but it's returning incorrect values.

commit 8141e617beb70f611988a336b98786b0be03e212
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu May 5 01:39:37 2022 -0400

    Update codegen.ml

commit 65a5188bc53fd6e18430b10192ab2b03c8567940
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu May 5 01:34:13 2022 -0400

    Update codegen.ml

commit 58dab46f1c20b7d5d29936008db76e3c5847849d
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed May 4 07:12:50 2022 -0400

    Update codegen.ml

commit 5634822440ecb5cff7e3ab11547ac213982eca3d
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue May 3 23:09:14 2022 -0400

    Add semantic check to bind.

commit c161dc4c720eb73e5263ea59da8a19145e61bb74
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue May 3 22:08:41 2022 -0400

    Add objects to codegen.ml - currently just act as structs.

commit 810646781d9e8d4960d4909a3af7c8229c74aedb
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon May 2 20:21:40 2022 -0400

    Comment out other list-related code in codegen.ml.

commit d911f9e2cf5a04f647b443694d01f8cb7530dbf9
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon May 2 20:19:56 2022 -0400

    Comment out incompelte code for lists in codegen.ml.
```

```
commit 7e41f9263c910b6a1e261cd5d39e51ef9ecbdec2
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon May 2 20:18:29 2022 -0400

    Fix missing opening comment symbol (* in codegen.ml.

commit edef96fe9aa8e8d31ee303c05dffb6d1cb520eca
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Sat Apr 30 21:10:42 2022 -0400

    Update codegen.ml

commit 8592dbbbc083bff7deb73cdd0e86442f23091fc2
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Sun Apr 24 08:02:06 2022 -0400

    Update codegen.ml

commit 96fe5afeec7b4e5b0b5f2d151e66503f93353c97
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Sun Apr 24 07:51:14 2022 -0400

    Update codegen.ml

commit babb96e85aaf32241f3894fcf0efdda5323b49ed
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 15:42:14 2022 -0400

    Prepare archive for submission.

commit 249aceed98b7512e700df99def811c63098ad37a
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 02:28:07 2022 -0400

    Update readme for the "Extended Testsuite" deliverable.

commit 7c9befa4062b8f0d30e6cedc171df10b663753af
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 02:11:33 2022 -0400

    Update expected output for parser tests due to changes in the ast printer.

commit 0870b66adca9a2d731b35538f7d23bb0a7c84e4e
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 02:00:55 2022 -0400

    Update the test target.

commit 39e0dc5a925606c754ccd3c0a1b310360e572db2
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 01:59:05 2022 -0400

    One new test and a bunch of fixed descriptions.

commit 1974001a62492a7450e7d0fe67ce04660a3117b1
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 01:41:00 2022 -0400

    Fix codegen for for-loops.

commit bb318140f18ef01f2cc04465eac008e0421388d0
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 01:40:30 2022 -0400

    Even more tests (continue, for, function call).
```

```
commit 3119a0f0654f41613ef72eb25be1e3aa72e4ac7a
Merge: a6ef87d 5046568
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Apr 20 01:11:05 2022 -0400

    Merge Chris's codegen with Randy's.

commit a6ef87d4758071199ab3da99056e288282dc3d6d
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Apr 20 01:09:26 2022 -0400

    Add for loops to codegen.ml; nested for loops do not work.

commit 50465682ec78ef07885f27476e3cf24e0e546c3f
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 00:50:04 2022 -0400

    Add tests for while, break and continue.

commit e5019493e1644337d9c41b298bb2443022fd723d
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 00:49:08 2022 -0400

    Fix up codegen for while & implement break and continue.

commit 8cb0abe2664f558300fcf217f387a3c3e9412c32
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Wed Apr 20 00:48:03 2022 -0400

    Improved diagnostics from the test script.

commit 04b1534125b762a4c025dd10e40f93cc28a232f3
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Apr 20 00:13:43 2022 -0400

    Add looping variables as locals in semant.ml.

commit 5b36bf16346874225049e82edf97f15603d16000
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Tue Apr 19 23:28:33 2022 -0400

    Add semantic check for break and continue and a relevant test.

commit f7ed39d6a94930b30faa461a2f948891d5d2f8be
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Tue Apr 19 23:02:14 2022 -0400

    Add expected output for new tests and extended tests target in Makefile.

commit 9c7f81571a304933e873701a04e44d3a865dd2de
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Tue Apr 19 22:57:11 2022 -0400

    Merge the two testsuites (expected output not ready yet).

commit 9d453f4eca797f845a2fc407996c32c2c85c7bb2
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Tue Apr 19 22:48:28 2022 -0400

    Implement codegen for the modulo operator. Add tests.

commit bb8b8a7e3b74edab8fb6f6e3df8d7ca9d3f40af1
Author: gfaline <gwenfedgar@gmail.com>
Date:   Tue Apr 19 22:28:17 2022 -0400
```

```
    Two more tests, while isnt ready yet

commit 9b1b44f446b506663e57749c01f18ab4922de597
Merge: 41e824a a230e00
Author: gfaline <gwenfedgar@gmail.com>
Date:   Tue Apr 19 22:07:59 2022 -0400

    Merge branch 'main' of https://github.com/gfaline/Compilers

commit 41e824aaa0de8f1ef6bef802b7e15d8752f50bea
Author: gfaline <gwenfedgar@gmail.com>
Date:   Tue Apr 19 22:07:48 2022 -0400

    else test

commit a230e00a34d99b7a247849ec64990d6db3fa8ecb
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Tue Apr 19 21:56:13 2022 -0400

    Make the compiler wrapper script fail cleanly.

commit 233f319411d42351b2fadc2a4cab7e738c6798f1
Author: gfaline <gwenfedgar@gmail.com>
Date:   Tue Apr 19 21:54:21 2022 -0400

    if elif else code, not testing yet

commit 3189f8a88bf93e5b638c35f0ff699b94c8678353
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Tue Apr 19 21:31:49 2022 -0400

    Implement type checking on return statements.

commit ce0838e98fea490ff3b079dfc1e272afdbab70ea
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Tue Apr 19 21:04:33 2022 -0400

    Eliminate all current warnings.

commit cad443c75731746d1e0af3381daaf436f7330b65
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 16:41:52 2022 -0400

    Add if/elif/else to codegen.ml.

commit e1d11171da188cb351d50a09fdcf31a8b9a06983
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 12:11:10 2022 -0400

    Add simple elif-less and else-less statements to codegen.ml.

commit b4e000e841e5896b9906ed70349beea9571f9f9a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 11:43:31 2022 -0400

    Remove commented-out code. Rename local function in SWhile stmt case.

commit 888cc5d9c887ac170d0889112c45d6ca23db5759
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 11:41:24 2022 -0400

    Add while loops to codegen.ml.

commit e04be72acc52f859175023b59b7e06b863e93357
```

```
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 11:15:38 2022 -0400

    Minor typographical changes to ast.ml and sast.ml for consistency.

commit b978550af08c785850c672292c45364f87ee3189
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 11:14:24 2022 -0400

    Add simple function calls to codegen.ml.

commit cd8d1506f9d05d1830d993d594b8a398c4bed1be
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 11:00:33 2022 -0400

    Add noexpr to codegen.ml.

commit 324b3b921410fb002e729d170d3784e15af15819
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:58:52 2022 -0400

    Add simple id evaluation to codegen.ml.

commit 71db814eed672a627e547bffb8080383dcf107c2
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:54:54 2022 -0400

    Add simple variable assignment to codegen.ml.

commit b28f316c6d7cd35d8c410e14d3d2ae5813685452
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:46:10 2022 -0400

    Simplify printing of bool literals in sast.ml.

commit 2524d58b3c86aec46d73125bd96f2de645a70485
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:45:42 2022 -0400

    Simplify printing of bool literals in ast.ml.

commit 5aa2014c52c8ac76565022de08da6f34d0d99bf2
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:43:34 2022 -0400

    Add parentheses expr to codegen.ml.

commit 8dcc243e2825d44b92575e88c31cdddbf7b18da8
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:41:16 2022 -0400

    Add simply unary operators for ints, float, and bools to codegen.ml.

commit d97488a9a483b97ebe72f8be2086768ce7bcc62f
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:34:06 2022 -0400

    Add simple binary operators for int/float to codegen.ml.

commit 4923f3740b43a64642eb964ee7acfaa76361ec9b
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:12:41 2022 -0400

    Whoops, boolean literals were already there.
```

```
commit fa0ed40eb853b3cf0b2aa841f68a4af123a6f400
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:10:06 2022 -0400

    Add boolean literals to codegen.ml.

commit 6fc874ef58245e7519c18ad80ea5b395476245c9
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 10:02:53 2022 -0400

    Add global variables to codegen.ml

commit 680ab71ed671dea9457899a4e0cf5df81d8dc4b5
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 09:55:52 2022 -0400

    Add simple unbind to sast.ml and semant.ml.

commit d949d5f38ec61e4b8f252e0adec8282e6221badb
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Apr 19 09:50:13 2022 -0400

    Add simple bind to sast.ml and semant.ml.

commit 9c458132dd8f9a655a29164e86103fbc0ab018ab
Merge: 57ce7b8 b503f1b
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 22:23:51 2022 -0400

    Merge codegen.ml with remote.

commit 57ce7b83dd948d95ad223c293361c160d89bb5e7
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 22:19:38 2022 -0400

    Add object property assignment to sast.ml and semant.ml.

commit 830e76616119e2a65246b14ce26e383023f275f1
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 22:09:13 2022 -0400

    Add object property dereferencing.

commit ddfb4eea621b2672e3acd3b2e13d82b4553e9921
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 19:54:24 2022 -0400

    Add comments to semant.ml.

commit 5c73a7fc4519dfcc363d544ed0db968eed133e5f
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 19:52:40 2022 -0400

    Simplify semant.ml.

commit 93485961f5bc9d810e90d79d24bb8deab06966f3
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 19:50:17 2022 -0400

    Add object definition to sast.ml and semant.ml. Simplify some printing things in the AST/SAST.

commit 45595c63d7ddcb3c536202a8176df88c4665b3e5
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 19:03:26 2022 -0400
```

```
    Add list indexing to sast.ml and semant.ml.

commit 9da4518dc94bc014a9ce2e6c4eabecfc0036aa6e
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 18:36:54 2022 -0400

    Add lists to sast.ml and semant.ml.

commit 798274a7fff99cf7295aae9142ff701950c697f0
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 17:37:20 2022 -0400

    Change OCaml representation of Propeller lists from list to array.

commit 526a245c23d59f36cd4df12c1c43fc293ca13262
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 16:57:41 2022 -0400

    Add break and continue to sast.ml and semant.ml.

commit 6675ae7e9475e5c5b3e19643eba8785da020ab43
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 16:52:49 2022 -0400

    Add simple while loops to sast and semant.ml.

commit 07815d38048ee19ab5125d4b90815084141a0f25
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 16:46:02 2022 -0400

    Add simple for loop to sast.ml and semant.ml.

commit e0dc68a4d1852dc2f678e2a43c61326ed7c66537
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 15:28:01 2022 -0400

    Add variable assignment to sast.ml and semant.ml.

commit 39ab29dd570edc5a7d01702b83822d625f054db0
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 18 15:22:06 2022 -0400

    Add ID evaluation to sast.ml and semant.ml.

commit 2d9edc7a6bdfe03e67749ecf10d3d13bcef54b5a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Apr 17 16:38:14 2022 -0400

    Fix typo in codegen.ml.

commit d5221d6ce9c82d16126972801f9e5dd1b3ea20f3
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Apr 17 16:37:12 2022 -0400

    Add floats to codegen.ml.

commit a21f6de6234477b3487fd367c00608d55063a6ae
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 23:30:17 2022 -0400

    Add return to sast and semant.

commit e0ebcb7ca5a306999a1145ac12dfde56fa406ae1
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 23:21:02 2022 -0400
```

```
    Add full if/elif/else support to sast.ml and semant.ml.

commit 23248059d8848643fbf5766293ef2eab96ad202a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 23:05:58 2022 -0400

    Apply code-reduction from ast.ml to sast.ml.

commit db52984db47fe6418e4d0fa5d8121d1f01ae5a5a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 22:51:57 2022 -0400

    Add function brace_wrap to reduce code in ast.ml.

commit 403d642f5a30dd4a3aab9f167f842a9dea362550
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 22:28:54 2022 -0400

    Simplify string_of_stmt_list If case.

commit be82bedc044b684be38fe2aeb62c8d630452002b
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 21:58:05 2022 -0400

    Add else-less if-elif stmts to semant.ml.

commit 4f67597b9580408f4c284659f8f1321a65d277dd
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 21:28:15 2022 -0400

    Add elif-less if-else stmts to sast.ml and semant.ml.

commit 84c5eaf63630de2105b2e187de7679b3e432978c
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 21:23:27 2022 -0400

    Fix printing for elif-less if-else stmt in ast.ml.

commit d5fdf7dc0860c9d47b585aed58aee56eeaf63f11
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Apr 14 21:20:09 2022 -0400

    Add elif-less and else-less if to sast.ml and semant.ml.

commit b152fb3d1720b76e38c72eac907510cd71c6779a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Apr 9 17:18:19 2022 -0400

    Add parentheses expression to sast.ml and semant.ml.

commit 6b85cc0134b33d426e8351902bccee01537e003e
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Apr 9 17:04:19 2022 -0400

    Add unary operators to sast.ml and semant.ml.

commit b503f1bd8dd850535e561eddf6a532e9a12295cf
Author: gfaline <gwenfedgar@gmail.com>
Date:   Sat Apr 9 13:45:37 2022 -0400

    typo... now all compiling

commit 31da9ae0967302de94f2b51d0f27335878a1032d
Author: gfaline <gwenfedgar@gmail.com>
```

```
Date:   Sat Apr 9 12:47:05 2022 -0400

    minor fix, sorry

commit f5e75fc94f71302427f2d21dccbc1457e5c09fe1
Author: gfaline <gwenfedgar@gmail.com>
Date:   Sat Apr 9 11:44:41 2022 -0400

    Added boolean operators and stuff

commit 8959fe64aaa6848c29ee69934ec6f9b72410fedd
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Apr 4 20:06:52 2022 -0400

    Add binary operators to sast.ml and semant.ml.

commit 93f670ca9e6c927203cfd610fd252e878ecc8682
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Mar 27 18:55:20 2022 -0400

    Move README.md to propeller folder since it's often zipped with other files.

commit 3724244bebe93ff10c2d1249bb724fa9e562233b
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Mar 27 18:54:13 2022 -0400

    Add submissions folder to keep track of past submissions. Initial commit contains hello-world.zip.

commit eddf52b52774b2fbc72f4463944112330cf7b677
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Mar 27 18:45:35 2022 -0400

    Add .exe and .out to .gitignore.

commit b084d142658b0c39cd0bc5260383b04a4d56c0e5
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Mar 27 18:44:23 2022 -0400

    Fix compilation issue by using more verbose useage of ocamlbuild from the MicroC README.

commit c754ff57abd30e147c8f597cc2267269d9a70821
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Mar 27 18:43:10 2022 -0400

    Add comment to README regarding description of tests in source files.

commit 764fc148226ade1dc65894af9b111677608aab0a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Mar 27 18:41:43 2022 -0400

    Add comments to each source file that describe the nature of the feature it tests.

commit 9a0584eed413f879cbaf2b748dc522f05959a0b7
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Fri Mar 25 13:31:30 2022 -0400

    Fix line endings for shell scripts.

commit 184ef20d369b2923a45723a4905338fa4df46eb3
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Fri Mar 25 13:24:30 2022 -0400

    Try fixing some EOL shenanigans.

commit 1822c9d90db74b6c83f4113c404895b98c0bdfe3
```

```
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Fri Mar 25 13:10:51 2022 -0400

    Fix an eggregious butchering of Gwen's name by Randy.

commit bb6ccda34ddf5a3aeda2aedf25933ec4c969e4d5
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Fri Mar 25 00:26:45 2022 -0400

    Update readme to reflect new testing infrastructure.

commit 737e734472442a02ebb4de0fc67c799f3d565ce7
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Fri Mar 25 00:26:31 2022 -0400

    Add option for preserving intermediate files.

commit e8a22f3d9af0f9b9ab35e887de1958b13a2d4243
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Fri Mar 25 00:26:07 2022 -0400

    Add a failure case for hello world tests.

commit eb370aab05cb6be84b0ffef7d71651693cde837f
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 23:56:11 2022 -0400

    Update Readme.

commit 73bdc5d2303305e36196347a922d3c6da11e6bb1
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 23:37:37 2022 -0400

    Add tests for hello world program. Add test targets to Makefile.

commit 650153a492550d3ac37d3c2f26b105faa4622c4a
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 23:17:21 2022 -0400

    Remove duplicate leftover files from old testing scripts.

commit 9dd4ed4a755c79ef3a988fa393ba2b28b74578a6
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 23:15:58 2022 -0400

    Rewritten the testing infrastructure.

commit ae47b5ad1a2be07a3e79e4fa5367e0b44db517c1
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 20:10:45 2022 -0400

    Add bare bones implementation of return. Fix print and integer literals.

commit c9637cdb6b08d19781ca5c465e65e761c7341d07
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 19:52:22 2022 -0400

    Eliminate all current warnings.

commit a342a28153c502978ebf94cbc58669ef85afcbc9
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 19:34:30 2022 -0400

    Add basic makefile for the compiler.
```

```
commit 4b5feb4e5f2091826c03bd18455aa7f59c6a82be
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 19:10:29 2022 -0400

    Ignore intermediate files in the compilation process.

commit 40e616955a06e97dde0f1a6007a05cf829b481a4
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 19:08:24 2022 -0400

    Add wrapper script for complete compiling pipeline.

commit 86933f3b6c40c5ea5cefa6acd9ecb2fd193f2582
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Thu Mar 24 18:35:02 2022 -0400

    Convert all files to LF line endings.

    ... and enforce LF on all text files in repository.

commit 3fba3a6e253a0910103beb11b447b2f9f70f47c5
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 17:01:43 2022 -0400

    Add str literals to sast.ml and semant.ml.

commit e2df34e84031560ccd42c2eed333488d9e277ad0
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 16:39:49 2022 -0400

    Add bool literals to sast.ml and semant.ml.

commit f957a997254eea5ee7cb63e0f0f7902f6517aff2
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 16:39:31 2022 -0400

    Add test.pr to gitignore.

commit 5226e9820cd747521a47e529cd69c3fcbccd8ef9
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 16:39:15 2022 -0400

    Remove test.pr (Randy's personal test file).

commit 28ca1d03a5701dffc6917cd5d391f5b8d81c875a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 16:29:36 2022 -0400

    Fix bug in string_of_sexpr. Add float literals to sast.ml and semant.ml.

commit 708d430fbabd6996ea62a4f78e3733f7e3695cff
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 16:03:13 2022 -0400

    Fix context name.

commit b865448e04f8540608a8a3e5c67f09c1aacb1303
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 15:50:52 2022 -0400

    Simple 'hello world' program can now be compiled.

commit 740144ca7e50f7b33364a576f310bbcc7098e772
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Mar 24 13:11:42 2022 -0400
```

```
    Add minimal codegen capabilities. LLVM IR can be printed.

commit 8acf2278dccfcf8d3b5efd43bd903fefda824ee1
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Mar 23 20:30:37 2022 -0400

    Add empty codegen.ml, sast.ml, semant.ml.

commit e45bc5ebe94b6af112fcc125e04fd9383fd5996b
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Mar 23 20:27:42 2022 -0400

    Rename scanner-parser folder to propeller.

commit 014eee06fef6b1e0f27247fac0a05068419db066
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Mar 23 20:26:55 2022 -0400

    Add other lrm files.

commit 5b3b91066ce83fafb5cf154e8c6e7171b773f871
Merge: 77d2e94 3b7c2bc
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Tue Mar 1 00:00:42 2022 -0500

    Merge branch 'main' of https://github.com/gfaline/Compilers

commit 3b7c2bc641fc4f45607145886e5a8730bb97c148
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Mon Feb 28 23:55:22 2022 -0500

    Fix errors in tex files.

commit 77d2e94a3c34d472af2db992b36d04f8c06fee86
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Mon Feb 28 23:57:19 2022 -0500

    Revert "Revert "Rewordings/corrections""

    This reverts commit de674e4116a3d7393d40c92afdeb81d1dc89421e.

commit de674e4116a3d7393d40c92afdeb81d1dc89421e
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Mon Feb 28 23:52:50 2022 -0500

    Revert "Rewordings/corrections"

    This reverts commit 54035b5976cea53fe2dad2fb0f27e72ce3165367.

commit 5fc6651e947a4c4f1c234af5459a888f32c137b4
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Mon Feb 28 23:52:41 2022 -0500

    crlf -> lf

commit 54035b5976cea53fe2dad2fb0f27e72ce3165367
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 28 23:48:30 2022 -0500

    Rewordings/corrections

commit 2daeefefd14c2523956f02d55e67c6a6addec5cb
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Mon Feb 28 22:26:37 2022 -0500
```

```
    Manually merged with the LRM project on overleaf.

commit 51c421ec52533f594c90fec93fd0cff708251b79
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Mon Feb 28 19:45:51 2022 -0500

    LRM updates for lists.

commit 79ec1a46a84d7427ee468e8d42e22ad387fc8407
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Mon Feb 28 18:41:46 2022 -0500

    Update LRM to reflect the new identifier rule.

commit f30500038a77d03d0887eb5f759250c7a4f0b603
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 28 13:18:55 2022 -0500

    Removed commented-out code.

commit cf0ab4f5f3e2c1fb06c17118b154a6b32f081a69
Merge: 8be36ab edc0a5a
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Mon Feb 28 13:17:04 2022 -0500

    Merge branch 'main' of https://github.com/gfaline/Compilers

commit 8be36ab703cf1e99289b8433483f974c19bc7831
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 28 13:16:29 2022 -0500

    Simplified regex for valid identifiers.

commit edc0a5afd4275b3aa7a75a9b0e5d6270cca2d52c
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Sun Feb 27 16:17:27 2022 -0500

    Fix typo in scanner.

commit 7f582bc4ccc3c14ca5345ab71d4fb329e53331a0
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Sun Feb 27 14:23:12 2022 -0500

    Add external keyword. Modify obj_decl to include extern field (bool).

commit 1f7b714f56b06aead33676c64c63512dea5425ad
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Sun Feb 27 13:13:43 2022 -0500

    Add declaration of variables of custom types.

commit 1e7b5d1aff77ebf7f8a9b63888f2d497ad5fe979
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Sun Feb 27 10:21:02 2022 -0500

    LRM initial edition.

commit 69fc853e513495d3fc78f4b5054108079a6889fa
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Fri Feb 25 11:39:13 2022 -0500

    Modified testall.sh to remove any CR from output files. Tests should now be platform-agnostic.

commit a0a2563874be651cf3ea61bbc1b303413f7bab9b
```

```
Author: Chris Xiong <chirs241097@gmail.com>
Date:   Fri Feb 25 10:37:53 2022 -0500

    purge useless files generated by macOS.

commit 59c65926e75112c44ecbc83765e62e0a7a21a427
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Fri Feb 25 10:37:33 2022 -0500

    Delete test-*.ll.

    Seems to have been committed by mistake - it's preventing me from pulling.

commit c334a0ecf8bb4fa5f44495fb4841613881ce407e
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 21:04:31 2022 -0500

    5th fixed

commit fb5285424ebfad23759949746feff5ac83084e68
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 20:51:09 2022 -0500

    Update .gitignore

    Just another testing generated file

commit 1021870f388169eababe1c7ec83a529b9375c9aa
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 20:49:49 2022 -0500

    Update testall.sh

    Didnt go back enough commits. Manually fixed it.

commit f2f802887c3831a6560ae5d7be9b7512cf6219f6
Merge: d18fba7 c3da99a
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 20:48:50 2022 -0500

    Merge pull request #9 from gfaline/testing_fix

    Testing fix

commit d18fba756f397a277f26515966deefe80373b975
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 20:45:47 2022 -0500

    Revert "Update testall.sh"

    This reverts commit dd5d63f990732d83cab4f9d6b588b68dbcd364c4.

commit 2b90a9d7134316f3ee70eeb66468456a6c36c144
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 20:44:15 2022 -0500

    Fixing the deletion stuff....

    Revert "Added newline to end end of string_of_program."

    This reverts commit f66cac8acb0b4a41d550dc7174fdd3a147182c30.

commit c3da99a63b8f9b5f8ab420005f7e4f20b34e7e09
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 20:41:02 2022 -0500
```

```
    Testing fix

commit dd5d63f990732d83cab4f9d6b588b68dbcd364c4
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 18:23:51 2022 -0500

    Update testall.sh

commit 895452f7a90062004f6d8720d646289a6ae6eaf2
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 17:39:39 2022 -0500

    Update testall.sh

commit f66cac8acb0b4a41d550dc7174fdd3a147182c30
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Feb 24 11:50:48 2022 -0500

    Added newline to end end of string_of_program.

commit bc2b3452165540124798c36e641fc35ffcf614b2
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Thu Feb 24 11:49:50 2022 -0500

    Removed runtests.py

commit e2b5ac6d43fa9c6a147b98592453cd87c8dcc72b
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 10:47:19 2022 -0500

    Tests

commit e3fb60db08ecb8384d9ab79a72ed311ac16b0e43
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 10:27:36 2022 -0500

    Added a requirements line

commit 75c00dd9801ed1dec18ec5f61c7e3740934787a8
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 10:26:25 2022 -0500

    Trying to do minor formatting.

commit f4ed1c68afe36afb93ec80252b4c464303220d34
Merge: 7db7f42 765974e
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 10:25:45 2022 -0500

    Merge pull request #7 from gfaline/testing

    Testing

commit 765974e9f302714ec42b7518d071d2cbffd1f780
Merge: ce7f6a6 7db7f42
Author: gfaline <gwenfedgar@gmail.com>
Date:   Thu Feb 24 10:25:30 2022 -0500

    Merge branch 'main' into testing

commit ce7f6a6cc3ea4904757f6e10b9670d7f453eadb1
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:11:06 2022 -0500
```

```
    Delete fail-test4.pr.diff

commit 77eceb3082257a31e01cecf29c149cc60003e032
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:10:56 2022 -0500

    Delete fail-test3.pr.diff

commit 7d50c4c14056f60d4f83cb59127e9767a04b4a72
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:10:43 2022 -0500

    Delete fail-test2.pr.diff

commit 806d1e708e16a2b1c1cfca8851b57a4c8e0868de
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:10:32 2022 -0500

    Delete fail-test1.pr.diff

commit 5bc4d1f136245d7d210be51bbe1fe7fe95e9f52b
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:10:20 2022 -0500

    Delete fail-test5.pr.diff

commit 7db7f42f7fa8ebf938f933b3aa698f5fc0b369c5
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:08:57 2022 -0500

    Delete fail-test1.pr.diff

commit 8a2e548582c151d2e9ad15e2f93af5638905826e
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:08:32 2022 -0500

    Add files via upload

commit 537d9e671771dc7628a3374fdeecd2dd4155e390
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:08:21 2022 -0500

    Add files via upload

commit ba49f4151b8cf2f9a9fb4edaa63e35d819460d94
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:07:40 2022 -0500

    Delete fail-test5.pr.diff

commit d36882e3ae0dd3cf384264b8d626774eedb8bd18
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:07:19 2022 -0500

    Delete fail-test4.pr.diff

commit 7d3e195c6a55659c35ce159eae5d48c348990401
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:07:11 2022 -0500

    Delete fail-test3.pr.diff

commit afea738ec41b7b658d4924cf0b9942e5847be308
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:06:53 2022 -0500
```

```
    Delete fail-test2.pr.diff

commit 949ded4929c4cef4c81f3e29f383488400c36b64
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:06:40 2022 -0500

    Delete fail-test1.pr.diff

commit 3210cc91325d1244126a989ddd9535448c1da352
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Thu Feb 24 03:05:20 2022 -0500

    Add files via upload

commit 5881a0138a26e2d3929de276babaacbb28a6073d
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Feb 23 23:25:23 2022 -0500

    Added description of differences between MicroC and Propeller, saying to assume that simple things were lifted from MicroC u

commit 3bfa1c728079e10abf8ce8889934fc02503dc29d
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Feb 23 22:29:17 2022 -0500

    Added document containing examples of Propeller syntax.

commit 88cb8e5ee7599775cba1f5a4474adb83297b3143
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Feb 23 22:28:43 2022 -0500

    Changed syntax of bind/unbind. Functions are now bound to object properties, not objects themselves. Fixed an issue where ar

commit 8ab9045a2357128fdb208e0c0c169f12ba66e36e
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Feb 23 21:33:33 2022 -0500

    Organized expr rules, expr type definition, and string_of_expr.

commit 1402546c74ae5ed0a472205dfff6c18d172ea863
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Wed Feb 23 21:20:56 2022 -0500

    Added list indexing.

commit df9142f4fbbfa556c1fed43ed5f2486e719f88a9
Merge: 642531a 2f2a7d4
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 20:35:26 2022 -0500

    Master merge

commit 2f2a7d45ebdd07b2e77b75cb12d27852eeac2455
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 20:33:47 2022 -0500

    More generated files being exculded

commit 642531ab8e1bbb8372bc448061887a86db1b7a66
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 20:32:48 2022 -0500

    Readme change

commit 4aebc59715a6acde4636ecdb14f795118fa8f851
```

```
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 20:18:52 2022 -0500

    it runs! Gotta check that it should be passing,ool

commit 443f2dd0eb0a7e44ce9f08cc66e954953b482977
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 19:14:58 2022 -0500

    Added an output file notation

commit 0754fa38d80fdb92705bde65979b7076f96e6f1e
Merge: cffe93f 3dbb3fb
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 19:13:35 2022 -0500

    merge fix

commit cffe93fada0c49d322016b2120b58c9b088c898f
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 19:11:49 2022 -0500

    To be the same, also readme done

commit 3dbb3fb17352a9dcde73aff2fb45f6d6b965db66
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:59:59 2022 -0500

    Update neg_test5.pr

commit 38d859c131f784e3b741cae34017d2b364a12aa6
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:54:22 2022 -0500

    Update pos_test5.pr

commit 2c78302f6c1b166a82aef160f0088052968a6b58
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:52:56 2022 -0500

    Update neg_test4.pr

commit a7a5518b795ebc5c07419d379d33aa7803eef625
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:52:27 2022 -0500

    Update neg_test3.pr

commit 128e11179804dc88f1bfe6b581b86c9be40d2575
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:51:59 2022 -0500

    Update neg_test2.pr

commit 5b358ab9d7d36e6bb72a8d559a15412f0f85765e
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:51:03 2022 -0500

    Update neg_test1.pr

commit 8abf45693ae9a7f39adaee5e490784b83c2b49b3
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:50:50 2022 -0500

    Update neg_test2.pr
```

```
commit d7985bc07f7b69f817fba898a46d28bff5c15d8e
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:50:31 2022 -0500

    Update neg_test3.pr

commit 6264fe82e1b7062d5c419c1420b590e998d0e2cf
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:49:51 2022 -0500

    Update neg_test3.pr

commit e6b2f92a41c5d78bc76e6ce584c256e75a8b5110
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:49:22 2022 -0500

    Update neg_test2.pr

commit 2e44c52dc99e56e975ee8a4e64aba231c47e205a
Author: ihamid01 <64386261+ihamid01@users.noreply.github.com>
Date:   Wed Feb 23 18:49:02 2022 -0500

    Update neg_test1.pr

commit b700ee69cbf80f00e557eaee1ed0aacc6a922db4
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 23 13:37:37 2022 -0500

    Testing changes, working on better system and outputs

commit c9bc772cf5af21a34e283575218e9737cede59e8
Author: gfaline <gwenfedgar@gmail.com>
Date:   Tue Feb 22 22:30:45 2022 -0500

    Trying out two testing methods. One like micro C and one not

commit 4ffa38b07288eebdb18f384cb1d53baa150e88a2
Author: gfaline <gwenfedgar@gmail.com>
Date:   Tue Feb 22 22:13:47 2022 -0500

    Added both for good measure

commit 33f3080904858a48757083209ae2e815fff7cb53
Author: gfaline <gwenfedgar@gmail.com>
Date:   Tue Feb 22 21:57:48 2022 -0500

    changed tests to fit microc format

commit bd1623f5b3838adbf7b3e0b975677f034b809887
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Feb 22 14:28:18 2022 -0500

    Added list literals.

commit fe5e3953c57778c25707d104dedcb50aec5f338b
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Feb 22 14:21:43 2022 -0500

    Added list type.

commit c636b45ef6b89bd0336718660a924be53156ebb7
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Feb 22 13:51:47 2022 -0500
```

```
    Added assignment to object properties.

commit fd2c31eff3718b7839fd6b0c83c165558d9697a0
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Feb 22 13:48:46 2022 -0500

    Added object property access (i.e. objname.fieldname as an expression).

commit f28da53754abda503ca34e28fd7e61a7455099df
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Feb 22 13:38:33 2022 -0500

    Removed whitespace.

commit 046e14f20c72b74c4c691ba2637b492eb49f559d
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Feb 22 13:37:37 2022 -0500

    Changed string_of_fdecl to print formals as (type1 name1, ...) instead of just (name1, ...).

commit 9b1dad4c88cdfaba3c0e6ff25c0c16485acd9565
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Tue Feb 22 13:23:09 2022 -0500

    Removed user-defined function join_strings because String.concat apparently does the same thing.

commit 642b98a19f61dd30f364041870280fb36b462ccb
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 22:28:59 2022 -0500

    Added bind/unbind statements.

commit 7bd250e0970cc9d148d443958c65a5916e8da0a1
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 22:15:22 2022 -0500

    Changed object definition term from obj to objdef to avoid ambiguous grammar. Removed distinction between ftype and vtype -

commit ae4eaabe1837edded1e389f3eca382bdf120867f
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 21:49:45 2022 -0500

    Removed obsolete helper functions from ast.ml. Fixed string_of_program to list vdecls, odecls, and fdecls in the order in wh

commit 0bbde646c01fb36114b4b37a857f750f938e2cf8
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 21:38:44 2022 -0500

    Added object declaration.

commit 122150daa47e530d1612919a1461c44606bf3d92
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 20:51:46 2022 -0500

    Added obj token to scanner and parser.

commit 4b525b2a0027f1f69463063914c75f82aee6110d
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 20:49:01 2022 -0500

    Removed commented-out code.

commit dc849b62c0ccde4a63ae0f6846afad642bef8576
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 15:24:25 2022 -0500
```

```
    Added break and continue statements.

commit fe747be4e76207519995f2b72d056cbe9eb112ac
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 15:14:42 2022 -0500

    Parentheses no longer required for exprs in if, elif, and while statements.

commit 5abc28e31f7dadb0a37f4dabf60310c45c22a713
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 15:06:45 2022 -0500

    Removed blocks; statements are now treated as lists with 1 or more statment. Braces now requires for if, elif, else, while,

commit 9f3f99b68b106014906a2ee8c49480ed7b211433
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 12:23:34 2022 -0500

    Made separate rules for expr and return statements.

commit 15945b24a7d29c1578e6516ad97be0be055252c8
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 12:05:25 2022 -0500

    Condenses ftyp and ftyp rules.

commit 11cefc424e004c63ca1bb2526a44246ca479c304
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 12:02:28 2022 -0500

    Made individual rules for for and while.

commit ebd1986a4be06101ff0daf9114326a78761f35de
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 11:13:04 2022 -0500

    Added elif.

commit 54fce49eebbe0afbac98988dd61915c1b997817e
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Mon Feb 21 09:19:18 2022 -0500

    Added if/else.

commit f2956003253c900c19cb8efe04d667904be3d796
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 23:48:14 2022 -0500

    Added simple if statement.

commit 0ebc6b8a1bbaceb53281c4a39469d67820ce9786
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 23:12:53 2022 -0500

    Added while.

commit f1c2109cb9951ddb0d4a56c98792c72106d1345a
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 19:23:58 2022 -0500

    Added statement blocks.

commit fa6f141edf766ab3e3a46c06120ac2a58d24fbc1
Author: Randy Price <edward.randolph.price@gmail.com>
```

```
Date:   Sun Feb 20 19:20:00 2022 -0500

    Added return statements.

commit 088e619675053815b3e147077dd91190f23851b6
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 18:41:58 2022 -0500

    Added negation of ints/floats.

commit 560dff7dc457b5813ebad49fde4b861a63e1d336
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 18:30:04 2022 -0500

    Added function calls.

commit 113c56321b6274f75bf934c25c6dcf61178d667d
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 18:20:06 2022 -0500

    Added logical NOT.

commit c66d8747ea4da2529230f8165b27e6f2ba45fd0c
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 17:42:42 2022 -0500

    Added binary logical operators.

commit d3662544ee03afba40beef9e732b8522e45e9e10
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 17:27:51 2022 -0500

    Added comparison operators.

commit abd20de456a73616b9177733ba66ba9dfc43ca3c
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 17:14:26 2022 -0500

    Added binary arithmetic operators, including modulus.

commit 11127789934e0f50e78a0484d215f6def09d3231
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 16:39:35 2022 -0500

    Expressions can now be wrapped in parentheses.

commit cb89f5de8ce4aca2d560c527c35f632764bf0060
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 16:33:51 2022 -0500

    Added assignment operator.

commit 9bbd1088b79408e18c55932d2b6f6f21e0e14717
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 16:28:29 2022 -0500

    Added simple string literals, whish are also expressions. Can't esscape single quotes yet - that ability will be added later

commit 7b9ecea917e4d45a16730f68a8118ccbd70f8540
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 15:40:07 2022 -0500

    Added boolean literals, which are also valid expresions.

commit 1b73af09f02647cb4484cf15bb73524963f63626
```

```
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 15:30:46 2022 -0500

    Added float literals, which also form valid expressions.

commit 2a993831f1025bb7c767778d22784064ea47aa42
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 15:25:01 2022 -0500

    Added integer literals, which also valid expressions.

commit 51e4d547cea5906cb00df95434e534e3509f44f2
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 15:18:28 2022 -0500

    Added expressions and statements. The only valid expression is an ID, and the only valid statment is an ID followed by a sem

commit 381e0f8e82b99e7834d3843dfc66f7a2d487a7b4
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 15:06:56 2022 -0500

    Local variables can now be declared inside functions.

commit 12e7eda1ee7c1f06b7c9c20cc570dc8f54e926b4
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 14:57:13 2022 -0500

    Bodyless functions can now be declared.

commit f1e16137f8edf9acb7b252776a67e5c49e85e7b8
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 13:09:06 2022 -0500

    Added void declaration type; cannot be used with variables, only with functions as a return type.

commit 49b06f1e3aaafef172c0c9bf4d4683f7720adf79
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 12:53:12 2022 -0500

    Fixed parsing error caused by commments at the end of a file; comments now terminate with either \n or EOF.

commit 063b24a89e78f928f13354073c6f5c36702ca8ff
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 12:51:22 2022 -0500

    Fixed typo in scanner.mll. Removed ~ as EOF token.

commit e6d6734a436e8e8a40b0dc4a07d8486f84b05bae
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 12:49:12 2022 -0500

    Comments erroneously began with // - now they begin with #.

commit b213b12f93d4660f8860a1caac5329debaa60eb9
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 12:47:29 2022 -0500

    Added bool, float, and str types for declaration.

commit 5f6965d401f4d22c041d24d7b387bd189daa74a1
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sun Feb 20 12:35:40 2022 -0500

    int variables can now be declared. A program type is now a list of integer bindings.
```

```
commit eee497e2b2eecdc58a942a07f5ca433d816cece4
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 17:24:08 2022 -0500

    Renamed TKN to ID to match MicroC and in-class examples.

commit 66ee2da9de2581b29706cfe2ae61cbc3c62499f1
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 17:12:27 2022 -0500

    Fixed bug in scanner.mll which permitted question marks in the middle of valid names.

commit 1696fad12288443ada75fe001c87abfffe432fde
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 17:00:56 2022 -0500

    Changed let () statment in toplevel.ml to accept an input .pr file for scanning/parsing/tree generation.

commit bae6d732b0add154e1ed7467045f0538fc0605b7
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 16:52:13 2022 -0500

    Scanner now throws specific exception for ill-formed names.

commit 19811e1f8b35c32cdb12cf5dfbda2f71909b7da6
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 15:34:05 2022 -0500

    Running a program now prints every token.

commit 6d5e2f65024ad73f0afef0532a4bba9cb0b1a941
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 15:14:17 2022 -0500

    scanner now discards single-line comments.

commit 4337d3e455f7c12f7577f2e1f6e197d5fa904545
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 15:09:26 2022 -0500

    All four files now interact with each other.

commit 56383520769739481f2f3d0da3adcc03846e3134
Author: Randy Price <edward.randolph.price@gmail.com>
Date:   Sat Feb 19 13:37:34 2022 -0500

    Added test directory with empty tests, and empty python script to execute them.

commit 2cae2a83d33b75cdd4a701d1b5ae0d0407172ead
Author: randyprice <79062334+randyprice@users.noreply.github.com>
Date:   Wed Feb 16 19:15:51 2022 -0500

    Added empty parser, scanner, AST, and top-level files.

commit 046f1efaad5061847436caf2e132ec34bf732270
Author: gfaline <gwenfedgar@gmail.com>
Date:   Wed Feb 9 13:09:58 2022 -0500

    Initial commit
```

9 Appendix B: Example Programs

9.1 Binding Demo

9.1.1 Propeller source code (binding.pr)

```
objdef Jumbo
{
  str name;
  int age;
  float gpa;
}

fn celebrate(int old, int new) -> void
{
  print(new);
}

fn init () -> int
{
  Jumbo jim;

  jim.name = 'Jim';
  jim.age  = 24;
  jim.gpa  = 3.73;

  bind(jim.age, celebrate);

  jim.age = 25;

  unbind(jim.age, celebrate);

  jim.age = 26;

  return 0;
}
```

9.1.2 Generated LLVM IR (binding.ll)

```
; ModuleID = 'Propeller'
source_filename = "Propeller"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%f\0A\00", align 1
@str = private unnamed_addr constant [4 x i8] c"Jim\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.4 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.5 = private unnamed_addr constant [4 x i8] c"%f\0A\00", align 1

declare i32 @printf(i8*, ...)

define i32 @init() {
entry:
  %jim = alloca { i8*, i32, double }, align 8
  %jim__name = getelementptr inbounds { i8*, i32, double }, { i8*, i32, double }* %jim, i32
    0, i32 0
```

```
  store i8* getelementptr inbounds ([4 x i8], [4 x i8]* @str, i32 0, i32 0), i8** %jim__name
    , align 8
  %jim__age = getelementptr inbounds { i8*, i32, double }, { i8*, i32, double }* %jim, i32
    0, i32 1
  store i32 24, i32* %jim__age, align 4
  %jim__gpa = getelementptr inbounds { i8*, i32, double }, { i8*, i32, double }* %jim, i32
    0, i32 2
  store double 3.730000e+00, double* %jim__gpa, align 8
  %jim__age1 = getelementptr inbounds { i8*, i32, double }, { i8*, i32, double }* %jim, i32
    0, i32 1
  store i32 25, i32* %jim__age1, align 4
  call void @celebrate(i32 25, i32 25)
  %jim__age2 = getelementptr inbounds { i8*, i32, double }, { i8*, i32, double }* %jim, i32
    0, i32 1
  store i32 26, i32* %jim__age2, align 4
  ret i32 0
}

define void @celebrate(i32 %old, i32 %new) {
entry:
  %old1 = alloca i32, align 4
  store i32 %old, i32* %old1, align 4
  %new2 = alloca i32, align 4
  store i32 %new, i32* %new2, align 4
  %new3 = load i32, i32* %new2, align 4
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @fmt
    .3, i32 0, i32 0), i32 %new3)
  ret void
}
```

9.1.3 Sensor Demo

9.1.4 Propeller source code (`sensor.pr`)

```
# compile with ./prc.sh -r sensor_linux sensor.pr

external objdef Sensor
{
  int temperature;
}

fn print_warning(int oldt, int t) -> void
{
  if (t > 60000) and (t != oldt)
  {
    prints('thermal zone sensor readout too high: ');
    print(t / 1000);
  }
}

fn init() -> int
{
  Sensor sensor;
  bind(sensor.temperature, print_warning);
  return 0;
}
```

9.1.5 Generated LLVM IR (`sensor.ll`)

```
; ModuleID = 'Propeller'
source_filename = "Propeller"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%f\0A\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.4 = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@fmt.5 = private unnamed_addr constant [4 x i8] c"%f\0A\00", align 1
@str = private unnamed_addr constant [39 x i8] c"thermal zone sensor readout too high: \00",
    align 1

declare i32 @printf(i8*, ...)

define i32 @init() {
entry:
  %sensor = alloca i32, align 4
  %created = call i32 @object_new_Sensor()
  store i32 %created, i32* %sensor, align 4
  %sensor1 = load i32, i32* %sensor, align 4
  call void @object_prop_bind_Sensor_temperature(i32 %sensor1, void (i32, i32)*
    @print_warning)
  ret i32 0
}

define void @print_warning(i32 %oldt, i32 %t) {
entry:
  %oldt1 = alloca i32, align 4
  store i32 %oldt, i32* %oldt1, align 4
  %t2 = alloca i32, align 4
  store i32 %t, i32* %t2, align 4
  %t3 = load i32, i32* %t2, align 4
  %tmp = icmp sgt i32 %t3, 60000
  %t4 = load i32, i32* %t2, align 4
  %oldt5 = load i32, i32* %oldt1, align 4
  %tmp6 = icmp ne i32 %t4, %oldt5
  %tmp7 = and i1 %tmp, %tmp6
  br i1 %tmp7, label %if, label %else

merge:                                            ; preds = %else, %if
  ret void

if:                                               ; preds = %entry
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt
    .4, i32 0, i32 0), i8* getelementptr inbounds ([39 x i8], [39 x i8]* @str, i32 0, i32 0)
    )
  %t8 = load i32, i32* %t2, align 4
  %tmp9 = sdiv i32 %t8, 1000
  %print10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
    @fmt.3, i32 0, i32 0), i32 %tmp9)
  br label %merge

else:                                             ; preds = %entry
  br label %merge
}

declare i32 @object_new_Sensor()

declare void @object_prop_bind_Sensor_temperature(i32, void (i32, i32)*)
```

9.1.6 C Runtime source code (sensor_linux.c)

```c
#include <stdio.h>
#include <unistd.h>

extern int init();

void(*boundf)(int,int) = NULL;
int oldt = 0;

int object_new_Sensor()
{
  return 0;
}

int object_prop_bind_Sensor_temperature(int oid, void(*f)(int,int))
{
  if (oid == 0)
    boundf = f;
}

int object_prop_unbind_Sensor_temperature(int oid, void(*f)(int,int))
{
  if (oid == 0 && boundf == f)
    boundf = NULL;
}

int object_prop_get_Sensor_temperature(int oid)
{
  if (oid == 0)
  {
    return oldt;
  }
  return 0;
}

int main()
{
  int ret = init();
  while (1)
  {
    FILE *f = fopen("/sys/class/thermal/thermal_zone0/temp", "r");
    int t;
    fscanf(f, "%d", &t);
    if (boundf) boundf(oldt, t);
    oldt = t;
    fclose(f);
    sleep(1);
  }
  return ret;
}
```

10 Appendix C: Testing Scripts

10.1 Main testing framework (`testutil.sh`)

```
TheCompiler=$(realpath $(dirname "$0")/../propeller.native)
TheWrapper=$(realpath $(dirname "$0")/../prc.sh)

# CheckExpectWReturnCode Exec ExpectedOutput ExpectedStderr ExpectedReturnCode
# Exec: command to execute
# ExpectedOutput: Reference output. Pass "IGNORED" to ignore all output.
# ExpectedStderr: Reference output of stderr. Pass "IGNORED" to ignore all output to stderr.
# ExpectedReturnCode: Expected return code. Pass "FAIL" to accept any non-zero return code.
CheckExpectWReturnCode() {
  RETCODE=0
  BADRET=0
  EXEC=$1
  ExpectedOutput=$2
  ExpectedStderr=$3
  ExpectedRet=$4
  OutputT=$(mktemp prtest.outt.XXXXX)
  StderrT=$(mktemp prtest.errt.XXXXX)
  Output=$(mktemp prtest.out.XXXXX)
  Stderr=$(mktemp prtest.err.XXXXX)

  eval "${EXEC} > ${OutputT} 2> ${StderrT}"
  Ret=$?
  tr -d '\015' < ${OutputT} > ${Output}
  tr -d '\015' < ${StderrT} > ${Stderr}

  [[ ${ExpectedOutput} == IGNORED ]] || diff -u ${ExpectedOutput} ${Output} || RETCODE=1
  [[ ${ExpectedStderr} == IGNORED ]] || diff -u ${ExpectedStderr} ${Stderr} || RETCODE=1

  if [[ ${ExpectedRet} == FAIL ]]; then
    [ $Ret -ne 0 ] || BADRET=1
  else
    [ $Ret -eq $ExpectedRet ] || BADRET=1
  fi

  if [[ $BADRET == 1 ]]; then
    RETCODE=1
    echo "unexpected return code $Ret"
  fi

  rm ${OutputT} ${StderrT} ${Output} ${Stderr}
  return $RETCODE
}

# CheckExpect Exec ExpectedOutput
# convenience function
CheckExpect() {
  EXEC=$1
  ExpectedOutput=$2
  CheckExpectWReturnCode "$EXEC" $ExpectedOutput IGNORED 0
}

# CheckFail Exec ExpectedStderr
# convenience function
CheckFail() {
  EXEC=$1
  ExpectedStderr=$2
  CheckExpectWReturnCode "$EXEC" IGNORED $ExpectedStderr FAIL
}


```

```
# CompiledCheckExpectWReturnCode Source ExpectedOutput ExpectedStderr ExpectedReturnCode
# convenience function. Compile the given propeller source program and run it with
# CheckExpectedWReturnCode
CompiledCheckExpectWReturnCode() {
  SRC=$1
  shift
  BASENAME=${SRC%.pr}
  EXEC=$BASENAME.out
  [[ $UNAME =~ (CYGWIN|MINGW|MSYS).* ]] && EXEC=$BASENAME.exe
  ${TheWrapper} $SRC
  CheckExpectWReturnCode "./$EXEC" $@
  RET=$?
  rm $EXEC
  return $RET
}

# CompiledCheckFail Exec ExpectedStderr
# convenience function
CompiledCheckExpect() {
  SRC=$1
  ExpectedOutput=$2
  CompiledCheckExpectWReturnCode $SRC $ExpectedOutput IGNORED 0
}

# CompiledCheckFail Exec ExpectedStderr
# convenience function
CompiledCheckFail() {
  SRC=$1
  ExpectedStderr=$2
  CompiledCheckExpectWReturnCode $SRC IGNORED $ExpectedStderr FAIL
}
```

10.2 The "Hello World" test suite (tests-hello.sh)

```
#!/usr/bin/env bash
source $(dirname "$0")/testutil.sh
cd $(dirname "$0")/hello

RET=0
CompiledCheckExpect hello.pr hello.exp || { echo "hello.pr failed"; RET=1; }
CompiledCheckExpectWReturnCode hello_ret.pr hello.exp IGNORED 5 || { echo "hello_ret.pr
    failed"; RET=1; }
CheckFail "$TheWrapper hello_bad.pr" IGNORED || { echo "hello_bad.pr failed"; RET=1; }
[ $RET -eq 0 ] && echo "all 3 tests passed"
exit $RET
```

10.3 The parser test suite (tests-parser.sh)

```
#!/usr/bin/env bash
source $(dirname "$0")/testutil.sh
cd $(dirname "$0")/parser

for i in test-*.pr
do
  out=${i%.pr}.exp
  if CheckExpect "$TheCompiler -a $i" $out; then
    echo "$i passed"
  else
    echo "$i failed"
  fi
done
```

```

for i in fail-*.pr
do
  err=${i%.pr}.err
  if CheckFail "$TheCompiler -a $i" $err; then
    echo "$i passed"
  else
    echo "$i failed"
  fi
done
```

10.4 The extended test suite (`tests-extended.sh`)

```
#!/usr/bin/env bash
source $(dirname "$0")/testutil.sh
cd $(dirname "$0")/exttest

failed=0

for i in test-*.pr
do
  out=${i%.pr}.exp
  if CompiledCheckExpect "$i" $out; then
    echo "$i passed"
  else
    echo "$i failed"
    failed=$(($failed+1))
  fi
done

for i in fail-*.pr
do
  err=${i%.pr}.err
  if CheckFail "$TheCompiler $i" $err; then
    echo "$i passed"
  else
    echo "$i failed"
    failed=$(($failed+1))
  fi
done

if [ $failed == 0 ]
then
  echo "All tests passed."
else
  echo "$failed test(s) failed."
fi
```

11 Appendix D: Full source code listing of the compiler

11.1 scanner.mll

```
(* Authors: Isra Ali, Gwendolyn Edgar, Randy Price, Chris Xiong *)
(* scanner for Propeller language *)

{ open Parser }

let letter = ['a'-'z' 'A'-'Z']
let digit  = ['0'-'9']
let alphnum = letter | digit
let identifier = letter '?'?
         | letter '_'? ( alphnum | alphnum '_')* alphnum '?'?

(* parse input *)
rule token = parse
  (* whitespace/comments *)
    [' ' '\t' '\r' '\n'] { token lexbuf }
  | '#' { comment lexbuf }
  (* syntactical symbols *)
  | '('       { LPAREN }
  | ')'       { RPAREN }
  | '{'       { LBRACE }
  | '}'       { RBRACE }
  | '['       { LBRCKT }
  | ']'       { RBRCKT }
  | ';'       { SEMI }
  | ','       { COMMA }
  | "fn"      { FN }
  | "->"      { ARROW }
  | '.'       { PERIOD }
  (* arithemtic operators *)
  | '+'       { PLUS }
  | '-'       { MINUS }
  | '*'       { TIMES }
  | '/'       { DIVIDE }
  | '%'       { MODULO }
  | '='       { ASSIGN }
  (* comparison operators *)
  | "=="      { EQ }
  | "!="      { NEQ }
  | '<'       { LT }
  | "<="      { LEQ }
  | ">"       { GT }
  | ">="      { GEQ }
  (* logical operators *)
  | "not"     { NOT }
  | "xor"     { XOR }
  | "and"     { AND }
  | "or"      { OR  }
  (* control flow *)
  | "for"     { FOR }
  | "from"    { FROM }
  | "to"      { TO }
  | "if"      { IF }
  | "elif"    { ELIF }
  | "else"    { ELSE }
  | "while"   { WHILE }
  | "break"   { BREAK }
  | "continue" { CONTINUE }
  | "return" { RETURN }
  (* Propeller stuff *)
  | "objdef" { OBJDEF }
```

```
  | "bind"   { BIND }
  | "unbind" { UNBIND }
  | "external" { EXTERNAL }
  (* primitive types *)
  | "obj"    { OBJ }
  | "int"    { INT }
  | "bool"   { BOOL }
  | "float"  { FLOAT }
  | "str"    { STR }
  | "void"   { VOID }
  | "list"   { LIST }
  (* literals *)
  | digit+ as x            { ILIT(int_of_string x)   }
  | digit+ '.' digit+ as x { FLIT(float_of_string x) }
  | "true"                 { BLIT(true) }
  | "false"                { BLIT(false) }
  | ''' [^''']*'''    as s { SLIT(s) }
  (* names *)
  | identifier   as id { ID(id) }
  | eof { EOF }
  | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

  and comment = parse
      '\n' { token lexbuf }
    | eof  { token lexbuf }
    | _    { comment lexbuf }
```

11.2 parser.mly

```
/* Authors: Isra Ali, Gwendolyn Edgar, Randy Price, Chris Xiong */
/* Ocamlyacc parser for Propeller */

%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA OBJDEF FN ARROW ASSIGN PLUS MINUS TIMES DIVIDE
    MODULO
%token NOT EQ NEQ LT LEQ GT GEQ XOR AND OR
%token EXTERNAL BIND UNBIND BREAK CONTINUE RETURN IF ELIF ELSE FOR FROM TO WHILE OBJ INT
   BOOL FLOAT STR VOID LIST
%token PERIOD
%token LBRCKT RBRCKT
%token <int> ILIT
%token <float> FLIT
%token <bool> BLIT
%token <string> SLIT
%token <string> ID
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc ELIF
%right ASSIGN
%left OR
%left AND
%left XOR
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT
```

```
33
34 %%
35
36 program:
37     decls EOF { $1 }
38
39 decls:
40     /* nothing */ { ([], [], []) }
41   | decls vdecl   { (($2 :: fst_trpl $1), snd_trpl $1, trd_trpl $1) }
42   | decls odecl   { (fst_trpl $1, ($2 :: snd_trpl $1), trd_trpl $1) }
43   | decls fdecl   { (fst_trpl $1, snd_trpl $1, ($2 :: trd_trpl $1)) }
44
45 odecl:
46     OBJDEF ID LBRACE vdecl_list RBRACE
47       { { oname = $2;
48           props = List.rev $4;
49           extern = false; } }
50   | EXTERNAL OBJDEF ID LBRACE vdecl_list RBRACE
51       { { oname = $3;
52           props = List.rev $5;
53           extern = true; } }
54
55 fdecl:
56   FN ID LPAREN formals_opt RPAREN ARROW typ LBRACE vdecl_list stmt_list RBRACE
57     { { typ     = $7;
58         fname   = $2;
59         formals = List.rev $4;
60         locals  = List.rev $9;
61         body    = List.rev $10 } }
62
63 formals_opt:
64     /* nothing */ { [] }
65   | formal_list   { $1 }
66
67 formal_list:
68     typ ID                   { [($1, $2)] }
69   | formal_list COMMA typ ID { ($3, $4) :: $1 }
70
71 typ:
72     INT   { Int }
73   | BOOL  { Bool }
74   | FLOAT { Float}
75   | STR   { Str }
76   | VOID  { Void }
77   | OBJ   { Obj }
78   | typ LIST { List($1) }
79
80 vdecl_list:
81     /* nothing */ { [] }
82   | vdecl_list vdecl { $2 :: $1 }
83
84 vdecl:
85     typ ID SEMI { ($1, $2) }
86   | ID ID SEMI { (Custom($1), $2) }
87
88 stmt_list:
89     // /* nothing */ { [] }
90     stmt { [$1] }
91   | stmt_list stmt { $2 :: $1 }
92
93 stmt:
94     expr_stmt   { $1 }
95   | return_stmt { $1 }
96   | if_stmt     { $1 }
97   | for_stmt    { $1 }
```

```
98    | while_stmt  { $1 }
99    | BREAK SEMI    { Break }
100   | CONTINUE SEMI { Continue }
101   | bind_stmt   { $1 }
102   | unbind_stmt { $1 }
103
104 bind_stmt:
105   BIND LPAREN ID PERIOD ID COMMA ID RPAREN SEMI { Bind ($3, $5, $7) }
106
107 unbind_stmt:
108   UNBIND LPAREN ID PERIOD ID COMMA ID RPAREN SEMI { Unbind ($3, $5, $7) }
109
110 expr_stmt:
111     expr SEMI { Expr($1) }
112
113 return_stmt:
114   | RETURN expr_opt SEMI { Return($2) }
115
116 if_stmt:
117     IF expr LBRACE stmt_list RBRACE elif_stmts else_stmt { If($2, List.rev $4, $6, $7) }
118
119 else_stmt:
120     %prec NOELSE { [] }
121   | ELSE LBRACE stmt_list RBRACE    { List.rev $3 }
122
123 elif_stmts:
124     /* nothing */ { [] }
125   | elif_stmts elif_stmt { $2 :: $1 }
126
127 elif_stmt:
128   ELIF expr LBRACE stmt_list RBRACE { ($2, List.rev $4) }
129
130 for_stmt:
131     FOR ID FROM expr TO expr LBRACE stmt_list RBRACE { For($2, $4, $6, List.rev $8) }
132
133 while_stmt:
134     WHILE expr LBRACE stmt_list RBRACE  { While($2, List.rev $4) }
135
136 expr_opt:
137     /* nothing */ { Noexpr }
138   | expr          { $1     }
139
140 expr:
141   // literals
142     ILIT { Iliteral($1) }
143   | FLIT { Fliteral($1) }
144   | BLIT { Bliteral($1) }
145   | SLIT { Sliteral($1) }
146   | LBRCKT args_list RBRCKT   { Lliteral(Array.of_list $2) }
147   // ID evaluation
148   | ID   { Id($1) }
149   | ID PERIOD ID %prec NOT    { Getprop($1, $3) }
150   // function call
151   | ID LPAREN args_opt RPAREN { Call($1, $3) }
152   // assignment
153   | ID ASSIGN expr            { Assign($1, $3) }
154   | ID PERIOD ID ASSIGN expr  { Setprop($1, $3, $5) }
155   // list indexing
156   | ID LBRCKT expr RBRCKT     { Index($1, $3) }
157   // arithmetic
158   | expr PLUS   expr          { Binop($1, Add, $3) }
159   | expr MINUS  expr          { Binop($1, Sub, $3) }
160   | expr TIMES  expr          { Binop($1, Mlt, $3) }
161   | expr DIVIDE expr          { Binop($1, Div, $3) }
162   | MINUS expr %prec NOT      { Unop(Neg, $2)  }
```

```
163   | expr MODULO expr          { Binop($1, Mod, $3) }
164   // comparison
165   | expr EQ     expr          { Binop($1, Eq,  $3) }
166   | expr NEQ    expr          { Binop($1, Neq, $3) }
167   | expr LT     expr          { Binop($1, Lt,  $3) }
168   | expr LEQ    expr          { Binop($1, Leq, $3) }
169   | expr GT     expr          { Binop($1, Gt,  $3) }
170   | expr GEQ    expr          { Binop($1, Geq, $3) }
171   // logical
172   | expr AND    expr          { Binop($1, And, $3) }
173   | expr XOR    expr          { Binop($1, Xor, $3) }
174   | expr OR     expr          { Binop($1, Or,  $3) }
175   | NOT  expr                 { Unop(Not, $2)      }
176   // other
177   | LPAREN expr RPAREN        { Parentheses($2) }
178
179 args_opt:
180     /* nothing */ { [] }
181   | args_list     { List.rev $1 }
182
183 args_list:
184     expr                { [$1]     }
185   | args_list COMMA expr { $3 :: $1 }
```

11.3 ast.ml

```
1  (* Authors: Isra Ali, Gwendolyn Edgar, Randy Price, Chris Xiong *)
2  let fst_trpl (a, _, _) = a
3  let snd_trpl (_, b, _) = b
4  let trd_trpl (_, _, c) = c
5
6  type binop =
7      Add
8    | Sub
9    | Mlt
10   | Div
11   | Mod
12   | Eq
13   | Neq
14   | Lt
15   | Leq
16   | Gt
17   | Geq
18   | Xor
19   | And
20   | Or
21
22 type unop =
23     Not
24   | Neg
25
26 type typ =
27     Int
28   | Bool
29   | Float
30   | Str
31   | Void
32   | Obj
33   | List of typ
34   | Custom of string
35
36 type bind = typ * string
37
```

```
38 type obj_decl = {
39   oname : string;
40   props : bind list;
41   extern: bool}
42
43 type expr =
44   (* literals *)
45     Iliteral of int
46   | Fliteral of float
47   | Bliteral of bool
48   | Sliteral of string
49   | Lliteral of expr array
50   (* function call *)
51   | Call of string * expr list
52   (* assignment *)
53   | Assign of string * expr
54   | Setprop of string * string * expr
55   (* ID evaluation*)
56   | Id of string
57   | Getprop of string * string
58   (* list indexing *)
59   | Index of string * expr
60   (* operators *)
61   | Binop of expr * binop * expr
62   | Unop of unop * expr
63   (* other *)
64   | Parentheses of expr
65   | Noexpr
66
67 type stmt =
68     Expr of expr
69   | Return of expr
70   | If of expr * stmt list * (expr * stmt list) list * stmt list
71   | For of string * expr * expr * stmt list
72   | While of expr * stmt list
73   | Break
74   | Continue
75   | Bind of string * string * string
76   | Unbind of string * string * string
77
78 type func_decl = {
79   typ : typ;
80   fname : string;
81   formals : bind list;
82   locals : bind list;
83   body : stmt list }
84
85 type program = bind list * obj_decl list * func_decl list
86
87 let string_of_binop = function
88     Add -> "+"
89   | Sub -> "-"
90   | Mlt -> "*"
91   | Div -> "/"
92   | Mod -> "%"
93   | Eq  -> "=="
94   | Neq -> "!="
95   | Lt  -> "<"
96   | Leq -> "<="
97   | Gt  -> ">"
98   | Geq -> ">="
99   | Xor -> "xor"
100  | And -> "and"
101  | Or  -> "or"
102
```

```
let string_of_unop = function
    Not -> "not"
  | Neg -> "-"

let rec string_of_typ = function
    Int   -> "int"
  | Bool  -> "bool"
  | Float -> "float"
  | Str   -> "str"
  | Void  -> "void"
  | Obj   -> "obj"
  | List(t) -> string_of_typ t ^ " list"
  | Custom(t) -> t

let rec string_of_expr = function
  (* literals *)
    Iliteral x -> string_of_int x
  | Fliteral x -> string_of_float x
  | Bliteral x -> if x then "true" else "false"
  | Sliteral x -> x
  | Lliteral xs -> "[" ^ String.concat ", " (Array.to_list (Array.map string_of_expr xs)) ^
    "]"
  (* function call *)
  | Call (f, es) -> f ^ "(" ^ String.concat ", " (List.map string_of_expr es) ^ ")"
  (* assignment *)
  | Assign (id, e) -> id ^ " = " ^ string_of_expr e
  | Setprop (o, p, e) -> o ^ "." ^ p ^ " = " ^ string_of_expr e
  (* ID evaluation *)
  | Id id -> id
  | Getprop (o, p) -> o ^ "." ^ p
  (* list indexing *)
  | Index (id, e) -> id ^ "[" ^ string_of_expr e ^ "]"
  (* operators*)
  | Binop (e1, op, e2) -> string_of_expr e1 ^ " " ^ string_of_binop op ^ " " ^
    string_of_expr e2
  | Unop (op, e) -> (match op with
      Not -> string_of_unop op ^ " (" ^ string_of_expr e ^ ")"
    | Neg -> string_of_unop op ^ "(" ^ string_of_expr e ^ ")")
  (* other *)
  | Parentheses e -> "(" ^ string_of_expr e ^ ")"
  | Noexpr -> ""


(* wrap stuff in curly braces *)
let brace_wrap s =
  "{\n" ^
  s ^ "\n" ^
  "}"

let rec string_of_stmt = function
    Expr(e) -> string_of_expr e ^ ";"
  | Return(e) -> (match e with
      Noexpr -> "return;"
    | _    -> "return " ^ string_of_expr e ^ ";")
  | If (e, s1, elifs, s2) ->
      let if_str =
        "if " ^ string_of_expr e ^ "\n" ^
        brace_wrap (String.concat "\n" (List.map string_of_stmt s1)) in
      let string_of_elif (elif_e, elif_s) =
        "elif " ^ string_of_expr elif_e ^ "\n" ^
        brace_wrap(String.concat "\n" (List.map string_of_stmt elif_s))
      in
      let elif_str = match elifs with
          [] -> ""
        | _  -> "\n" ^
```

```
                String.concat "\n" (List.map string_of_elif elifs) in
      let else_str = match s2 with
          [] -> ""
        | _  -> "\n" ^
                "else\n" ^
                brace_wrap(String.concat "\n" (List.map string_of_stmt s2)) in
      if_str ^ elif_str ^ else_str
  | While(e, s) ->
      "while " ^ string_of_expr e ^ "\n" ^
      brace_wrap (String.concat "\n" (List.map string_of_stmt s))
  | For(id, e1, e2, s) ->
      "for " ^ id ^ " from " ^ string_of_expr e1 ^ " to " ^ string_of_expr e2 ^ "\n" ^
      brace_wrap (String.concat "\n" (List.map string_of_stmt s))
  | Break -> "break;"
  | Continue -> "continue;"
  | Bind(o, p, f) -> "bind( " ^ o ^ "." ^ p ^ ", " ^ f ^ ");"
  | Unbind(o, p, f) -> "unbind( " ^ o ^ "." ^ p ^", " ^ f ^ ");"

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";"

let string_of_odecl odecl =
  if odecl.extern then
    "external objdef " ^ odecl.oname ^ "\n" ^
    brace_wrap (String.concat "\n" (List.map string_of_vdecl odecl.props))
  else
    "objdef " ^ odecl.oname ^ "\n" ^
    brace_wrap (String.concat "\n" (List.map string_of_vdecl odecl.props))

let string_of_formal (t, id) = string_of_typ t ^ " " ^ id

let string_of_fdecl fdecl =
  "fn " ^ fdecl.fname ^ "("  ^ String.concat ", " (List.map string_of_formal fdecl.formals)
    ^ ") -> " ^ string_of_typ fdecl.typ ^ "\n" ^
  brace_wrap ((String.concat "\n" (List.rev (List.map string_of_vdecl fdecl.locals))) ^ "\n\
    n" ^
               String.concat "\n" (List.map string_of_stmt fdecl.body))

let string_of_program (vdecls, odecls, fdecls) =
  String.trim
  (String.concat "\n" (List.rev (List.map string_of_vdecl vdecls)) ^ "\n" ^
  String.concat "\n\n" (List.rev (List.map string_of_odecl odecls)) ^ "\n" ^
  String.concat "\n\n" (List.rev (List.map string_of_fdecl fdecls))) ^ "\n"
```

11.4 codegen.ml

```
(* Authors: Isra Ali, Gwendolyn Edgar, Randy Price, Chris Xiong *)
module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

let translate (globals, objects, functions) =
  let context = L.global_context () in

  let i32_t      = L.i32_type      context
  and i8_t       = L.i8_type       context
  and i1_t       = L.i1_type       context
  and float_t    = L.double_type   context
  and void_t     = L.void_type     context
  and the_module = L.create_module context "Propeller" in

  (* let objdef_strs = String.concat "\n" (List.map (fun o -> o.soname) objects) in *)
```

```
19
20  let is_external = function
21      A.Custom t ->
22        let objdef = List.find (fun o -> o.soname = t) objects in
23        objdef.sextern
24    | _ -> false
25  in
26
27  let rec ltype_of_typ = function
28      A.Int   -> i32_t
29    | A.Float -> float_t
30    | A.Bool  -> i1_t
31    | A.Str ->  L.pointer_type (L.i8_type (context))
32    | A.Void  -> void_t
33    | A.Custom t ->
34        let objdef = List.find (fun o -> o.soname = t) objects in
35        let ptys = List.map fst objdef.sprops in
36        let ltys = Array.of_list (List.map ltype_of_typ ptys) in
37        if objdef.sextern then i32_t else L.struct_type context ltys
38    | A.List(t) -> L.pointer_type (ltype_of_typ t)
39    | _       -> i32_t
40  in
41
42  (* indices for struct getelementpointer *)
43  let get_obj_gep_idx o p =
44    let obj_geps =
45      let add_obj m odecl =
46        let rec build_pmap n = function
47            []         -> StringMap.empty
48          | (_, p)::ps -> StringMap.add p n (build_pmap (n + 1) ps)
49        in
50        let pmap = build_pmap 0 odecl.sprops in
51        StringMap.add odecl.soname pmap m
52      in
53      List.fold_left add_obj StringMap.empty objects in
54    StringMap.find p (StringMap.find o obj_geps)
55  in
56
57  (* globals *)
58  let global_vars : L.llvalue StringMap.t =
59    let global_var m (t, n) =
60      let init = match t with
61          A.Float -> L.const_float (ltype_of_typ t) 0.0
62        | _ -> L.const_int (ltype_of_typ t) 0 in
63      StringMap.add n (L.define_global n init the_module) m
64    in
65    List.fold_left global_var StringMap.empty globals in
66
67  (* functions *)
68  let print_t : L.lltype = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
69  let print_func : L.llvalue = L.declare_function "printf" print_t the_module in
70
71  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
72    let function_decl m fdecl =
73      let name = fdecl.sfname
74      and formal_types = Array.of_list (List.map (fun (t, _) -> ltype_of_typ t) fdecl.
    sformals) in
75      let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
76      StringMap.add name (L.define_function name ftype the_module, fdecl) m
77    in
78    List.fold_left function_decl StringMap.empty functions in
79
80  (* global symbols *)
81  let gsyms = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
82                             StringMap.empty globals in
```

```
83
84
85  (* ================ FUNCTION BODY ================ *)
86  let build_function_body fdecl =
87
88    let (the_function, _) = StringMap.find fdecl.sfname function_decls in
89    let builder = L.builder_at_end context (L.entry_block the_function) in
90
91    let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
92    let str_format_str = L.build_global_stringptr "%s" "fmt" builder in
93    let float_format_str = L.build_global_stringptr "%f\n" "fmt" builder in
94
95    let local_vars =
96      let add_formal m (t, n) p =
97        let () = L.set_value_name n p in
98        let local = L.build_alloca (ltype_of_typ t) n builder in
99        let _  = L.build_store p local builder in
100       StringMap.add n local m
101     in
102     let add_local m (t, n) =
103       let local_var = L.build_alloca (ltype_of_typ t) n builder
104       in StringMap.add n local_var m
105     in
106     let formals = List.fold_left2 add_formal StringMap.empty fdecl.sformals
107                                   (Array.to_list (L.params the_function)) in
108     List.fold_left add_local formals fdecl.slocals in
109
110   let lookup n =
111     try  StringMap.find n local_vars
112     with Not_found -> StringMap.find n global_vars
113   in
114
115   (* type of symbols *)
116   let fsyms = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
117                              StringMap.empty fdecl.sformals in
118   let lsyms = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
119                              StringMap.empty fdecl.slocals in
120
121   let type_of_identifier id =
122     try StringMap.find id lsyms
123     with Not_found ->
124       try StringMap.find id fsyms
125       with Not_found ->
126         try StringMap.find id gsyms
127         with Not_found -> raise (Failure ("Internal error - undefined identifier " ^ id))
128   in
129
130   let type_of_prop o p =
131     let objpropt =
132       let add_obj m odecl =
133         let rec build_tmap = function
134             []          -> StringMap.empty
135           | (t, p)::ps -> StringMap.add p t (build_tmap ps)
136         in
137         let tmap = build_tmap odecl.sprops in
138         StringMap.add odecl.soname tmap m
139       in
140       List.fold_left add_obj StringMap.empty objects in
141     StringMap.find p (StringMap.find o objpropt)
142   in
143
144   (* get name of object type (objdef <type>) *)
145   let type_of_obj o = match type_of_identifier o with
146       A.Custom t -> t
147     | _ -> raise (Failure (o ^ " is not an object"))
```

```
148     in
149
150     let extcreatef_t : L.lltype = L.function_type i32_t [| |] in
151
152     let build_extern_local_init builder =
153       let init_extern_local builder (t, n) =
154         if is_external (type_of_identifier n) then
155           match t with
156               A.Custom t ->
157                 let create_func : L.llvalue = L.declare_function ("object_new_" ^ t)
    extcreatef_t the_module in
158                 let r = L.build_call create_func [| |] "created" builder in
159                 let _ = L.build_store r (StringMap.find n local_vars) builder in
160               builder
161             | _ -> raise (Failure ("internal codegen error"))
162         else builder
163       in List.fold_left init_extern_local builder fdecl.slocals
164     in
165
166     (* ================ PROPERTY BINDING MAP ================ *)
167     (* map of properties and their bound functions : string -> string list
168        <var_name>__<prop_name> -> [<function names>] *)
169     let bind_map =
170       (* names of local objects *)
171       let local_obj_names =
172         let is_obj = function
173             (A.Custom _, _) -> true
174           | _ -> false
175         in
176         let local_obj_vars = List.filter is_obj fdecl.slocals in
177         List.map snd local_obj_vars in
178
179       (* make keys for property-function bindings map *)
180       let bind_map_keys =
181         let rec make_bind_map_keys = function
182             [] -> []
183           | o::os -> let props =
184                        let get_prop_names_of_var v =
185                          let objdef = type_of_obj v in
186                          let odecl = try List.find (fun od -> od.soname = objdef) objects
187                                      with Not_found -> raise (Failure "can't find object")
    in
188                          let oprops = odecl.sprops in
189                          List.map snd oprops
190                        in
191                        get_prop_names_of_var o in
192                        let make_key p =
193                          o ^ "__" ^ p
194                        in
195                        (List.map make_key props) @ (make_bind_map_keys os)
196         in
197         make_bind_map_keys local_obj_names in
198
199       (* let bind_map_keys = make_bind_map_keys local_obj_names in *)
200       let make_empty_lists m k =
201         StringMap.add k [] m
202       in
203       ref (List.fold_left make_empty_lists StringMap.empty bind_map_keys) in
204
205     (* add binding to property*)
206     let add_obj_bind o p f =
207       let k = (o ^ "__" ^ p) in
208       let fs = StringMap.find k !bind_map in
209       if List.mem f fs
210       then raise (Failure ("function " ^ f ^ " is already bound to " ^ o ^ "." ^ p))
```

```
211       else
212         let new_fs = f :: fs in
213         let new_m = StringMap.add k new_fs !bind_map in
214         bind_map := new_m
215     in
216
217     (* remove binding from property *)
218     let rem_obj_bind o p f =
219       let k = (o ^ "__" ^ p) in
220       let fs = StringMap.find k !bind_map in
221       if List.mem f fs
222       then
223         let new_fs = List.filter (fun n -> n <> f) fs in
224         let new_m = StringMap.add k new_fs !bind_map in
225         bind_map := new_m
226       else raise (Failure ("function " ^ f ^ " is not bound to " ^ o ^ "." ^ p))
227     in
228
229     (* get a property's bound functions *)
230     let get_bound_funcs o p =
231       let k = (o ^ "__" ^ p) in
232       StringMap.find k !bind_map
233     in
234
235     (* ================ EXPRESSION BUILDER ================ *)
236     let rec expr builder ((_, e) : sexpr) = match e with
237         SIliteral x -> L.const_int i32_t x
238       | SFliteral x -> L.const_float float_t x
239       | SBliteral x -> L.const_int i1_t (if x then 1 else 0)
240       | SSliteral x ->  L.build_global_stringptr x "str" builder
241       | SLliteral xs ->
242         let (x, _) = Array.get xs 0 in
243         let allocate = L.build_array_alloca (ltype_of_typ x) (L.const_int i32_t (Array.
    length xs)) "tmp_list" builder in
244         let build_list x i arr =
245           let gep_ptr = L.build_gep arr [| L.const_int i32_t i |] "list" builder in
246           let _ = L.build_store (expr builder x) gep_ptr builder in
247           i + 1
248         in
249         let _ =  Array.fold_left (fun y el -> build_list el y allocate) 0 (Array.of_list (
    List.rev (Array.to_list xs))) in
250         allocate
251       | SCall (f, es) -> (match (f, es) with
252             ("print", [e]) | ("printb", [e]) ->
253               L.build_call print_func [| int_format_str ; (expr builder e) |] "print"
    builder
254           | ("prints", [e]) -> L.build_call print_func [| str_format_str ; (expr builder e)
    |] "print" builder
255           | ("printf", [e]) -> L.build_call print_func [| float_format_str ; (expr builder e
    ) |] "print" builder
256           | _ -> let (fdef, fdecl) = StringMap.find f function_decls in
257                  let lles = List.rev (List.map (expr builder) (List.rev es)) in
258                  let result = (match fdecl.styp with
259                                  A.Void -> ""
260                                | _ -> f ^ "_result") in
261                  L.build_call fdef (Array.of_list lles) result builder)
262       | SAssign (id, e) ->
263         let e' = expr builder e in
264         let _ = L.build_store e' (lookup id) builder in
265         e'
266       | SSetprop (o, p, e) ->
267         if is_external (type_of_identifier o)
268         then
269           raise (Failure "Property assignment to external objects is unimplemented")
270         else
```

```
271         let e' = expr builder e in
272         let objtype = type_of_obj o in
273         let idx = get_obj_gep_idx objtype p in
274         let id = (o ^ "__" ^ p) in
275         let tmp = L.build_struct_gep (lookup o) idx id builder in
276         let _ = L.build_store e' tmp builder in
277         let fs = get_bound_funcs o p in
278         let call_bound_func f =
279           expr builder (A.Void, SCall(f, [e; e]))
280         in
281         let _ = List.map call_bound_func fs in
282         e'
283     | SId id -> L.build_load (lookup id) id builder
284     | SGetprop (o, p) ->
285       if is_external (type_of_identifier o)
286       then
287         let oty = type_of_obj o in
288         let pty = type_of_prop (type_of_obj o) p in
289         let extgetf_t : L.lltype  = L.function_type (ltype_of_typ pty) [| i32_t |] in
290         let get_func  : L.llvalue = L.declare_function ("object_prop_get_" ^ oty ^ "_" ^ p
    ) extgetf_t the_module in
291         L.build_call get_func [| (lookup o) |] "get_result" builder
292       else
293         let objtype = type_of_obj o in
294         let idx = get_obj_gep_idx objtype p in
295         let tmp = L.build_struct_gep (lookup o) idx (o ^ "__" ^ p) builder in
296         L.build_load tmp (o ^ "__" ^ p) builder
297     | SIndex (id, e) ->
298       let id' = lookup id  in
299       let indx = expr builder e in
300       let head_ptr = L.build_load id' id builder in
301       let elem_ptr  = L.build_gep head_ptr [|indx|] "p" builder in
302       L.build_load elem_ptr "tmp" builder
303     | SBinop (e1, op, e2) ->
304         let (t, _) = e1
305         and e1' = expr builder e1
306         and e2' = expr builder e2 in
307         let instr = (match t with
308              A.Int -> (match op with
309                          A.Add -> L.build_add
310                        | A.Sub -> L.build_sub
311                        | A.Mlt -> L.build_mul
312                        | A.Div -> L.build_sdiv
313                        | A.Mod -> L.build_srem
314                        | A.Eq  -> L.build_icmp L.Icmp.Eq
315                        | A.Neq -> L.build_icmp L.Icmp.Ne
316                        | A.Lt  -> L.build_icmp L.Icmp.Slt
317                        | A.Leq -> L.build_icmp L.Icmp.Sle
318                        | A.Gt  -> L.build_icmp L.Icmp.Sgt
319                        | A.Geq -> L.build_icmp L.Icmp.Sge
320                        | _ -> raise (Failure "internal error - bad int operator"))
321            | A.Float -> (match op with
322                            A.Add -> L.build_fadd
323                          | A.Sub -> L.build_fsub
324                          | A.Mlt -> L.build_fmul
325                          | A.Div -> L.build_fdiv
326                          | A.Eq  -> L.build_fcmp L.Fcmp.Oeq
327                          | A.Neq -> L.build_fcmp L.Fcmp.One
328                          | A.Lt  -> L.build_fcmp L.Fcmp.Olt
329                          | A.Leq -> L.build_fcmp L.Fcmp.Ole
330                          | A.Gt  -> L.build_fcmp L.Fcmp.Ogt
331                          | A.Geq -> L.build_fcmp L.Fcmp.Oge
332                          | _ -> raise (Failure "internal error - bad float operator"))
333             | A.Bool -> (match op with
334                            A.Eq  -> L.build_icmp L.Icmp.Eq
```

```
                            | A.Neq -> L.build_icmp L.Icmp.Ne
                            | A.And -> L.build_and
                            | A.Or  -> L.build_or
                            | A.Xor -> raise (Failure "internal error - bad float operator"
    )
                            | _ -> raise (Failure "internal error - bad bool operator"))
            | _ -> raise (Failure ("internal error - bad binary operator type"))) in
          instr e1' e2' "tmp" builder
      | SUnop (op, e) ->
          let (t, _) = e in
          let e' = expr builder e in
          let instr = (match op with
                          A.Neg when t = A.Float -> L.build_fneg
                        | A.Neg when t = A.Int   -> L.build_neg
                        | A.Not when t = A.Bool  -> L.build_not
                        | _ -> raise (Failure "bad unary operator/operand")) in
          instr e' "tmp" builder
      | SParentheses e -> expr builder e
      | SNoexpr -> L.const_int i32_t 0
    in

    let add_terminal builder instr = match L.block_terminator (L.insertion_block builder)
    with
        Some _ -> ()
      | None -> ignore (instr builder)
    in

    (* ================ STATEMENT BUILDER ================ *)
    let rec stmt loop_start loop_after builder = function
        SExpr e -> let _ = expr builder e in builder
      | SReturn e ->
          let _ = match fdecl.styp with
                      A.Void -> L.build_ret_void builder
                    | _      -> L.build_ret (expr builder e) builder in
          builder
      | SIf (e, s1, elifs, s2) ->

          (* convert elifs to nested SIf *)
          let s2' = (match elifs with
              [] -> s2
            | _  ->
                let rec transform_elifs = function
                    [(elif_e, elif_s)]     -> SIf(elif_e, elif_s, [], s2)
                  | (elif_e, elif_s) :: es  -> SIf(elif_e, elif_s, [], [transform_elifs es])
                  | [] -> raise (Failure "semant internal error")
                in
                [transform_elifs elifs]) in

          (* merge block, branch to merge block instruction *)
          let merge_bb = L.append_block context "merge" the_function in
          let branch_instr = L.build_br merge_bb in

          (* add instruction builders for a list of statements to the end of a basic block
    *)
          let build_bb_stmts bb s =
            let build_bb_stmt st =
              stmt loop_start loop_after (L.builder_at_end context bb) st
            in
            List.map build_bb_stmt s
          in

          (* add terminal to end of list of statements *)
          let rec terminate = function
              [] -> ()
            | (t :: []) -> add_terminal t branch_instr
```

```
397             | (_ :: ts) -> terminate ts
398           in
399
400           (* if *)
401           let if_bool = expr builder e in
402           let if_bb = L.append_block context "if" the_function in
403           let if_builders = build_bb_stmts if_bb s1 in
404           let () = terminate if_builders in
405
406           (* else *)
407           let else_bb = L.append_block context "else" the_function in
408           let else_builders = (match s2' with
409               [] -> [stmt loop_start loop_after (L.builder_at_end context else_bb) (SExpr (A
    .Int, SNoexpr))]
410             | _ ->  build_bb_stmts else_bb s2') in
411           let () = terminate else_builders in
412
413           (* build stuff *)
414           let _ = L.build_cond_br if_bool if_bb else_bb builder in
415           L.builder_at_end context merge_bb
416
417       | SWhile (e, s) ->
418           let e_bb = L.append_block context "while" the_function in
419           let _    = L.build_br e_bb builder in
420           let s_bb = L.append_block context "while_body" the_function in
421           let merge_bb = L.append_block context "merge" the_function in
422           let while_builder = List.fold_left (stmt (Some e_bb) (Some merge_bb))
423                               (L.builder_at_end context s_bb) s in
424           let () = add_terminal while_builder (L.build_br e_bb) in
425           let e_builder = L.builder_at_end context e_bb in
426           let bool_val = expr e_builder e in
427           let _ = L.build_cond_br bool_val s_bb merge_bb e_builder in
428           L.builder_at_end context merge_bb
429       | SFor (id, e1, e2, s) ->
430           let b = stmt loop_start loop_after builder (SExpr (A.Int, SAssign(id, e1))) in
431           let id_sexpr = (A.Int, SId id) in
432           let cmp_sexpr = (A.Bool, SBinop(id_sexpr, A.Leq, e2)) in
433           let int1_sexpr = (A.Int, SIliteral 1) in
434           let add_sexpr = (A.Int, SBinop (id_sexpr, A.Add, int1_sexpr)) in
435           let inc_sexpr = (A.Int, SAssign(id, add_sexpr)) in
436           let inc_stmt = SExpr inc_sexpr in
437           let while_stmts = s @ [inc_stmt] in
438           stmt loop_start loop_after b (SWhile (cmp_sexpr, while_stmts))
439       | SContinue ->
440           (match loop_start with
441             (Some bb) -> let _ = L.build_br bb builder in
442                          builder
443           | None -> raise (Failure "semant internal error"))
444       | SBreak ->
445           (match loop_after with
446             (Some bb) -> let _ = L.build_br bb builder in
447                          builder
448           | None -> raise (Failure "semant internal error"))
449       | SBind(o, p, f) ->
450         if is_external (type_of_identifier o)
451         then
452           let oty = type_of_obj o in
453           let pty = type_of_prop (type_of_obj o) p in
454           let extboundfuncp_t : L.lltype = L.pointer_type (L.function_type void_t [|
    ltype_of_typ pty; ltype_of_typ pty |]) in
455           let extbindf_t     : L.lltype = L.function_type void_t [| i32_t; extboundfuncp_t
    |] in
456           let bind_func : L.llvalue = L.declare_function ("object_prop_bind_" ^ oty ^ "_" ^
    p) extbindf_t the_module in
457           let ov = L.build_load (lookup o) o builder in
```

```
458           let _ = L.build_call bind_func [| ov; fst (StringMap.find f function_decls) |] ""
    builder in
459           builder
460         else let _ = add_obj_bind o p f in builder
461       | SUnbind(o, p, f) ->
462         if is_external (type_of_identifier o)
463         then
464           let oty = type_of_obj o in
465           let pty = type_of_prop (type_of_obj o) p in
466           let extboundfuncp_t : L.lltype = L.pointer_type (L.function_type void_t [|
    ltype_of_typ pty; ltype_of_typ pty |]) in
467           let extbindf_t      : L.lltype = L.function_type void_t [| i32_t; extboundfuncp_t
    |] in
468           let unbind_func : L.llvalue = L.declare_function ("object_prop_unbind_" ^ oty ^ "_
    " ^ p) extbindf_t the_module in
469           let ov = L.build_load (lookup o) o builder in
470           let _ = L.build_call unbind_func [| ov; fst (StringMap.find f function_decls) |] "
    " builder in
471           builder
472         else let _ = rem_obj_bind o p f in builder
473     in
474
475     let b = build_extern_local_init builder in
476     let builder = List.fold_left (stmt None None) b fdecl.sbody in
477
478     add_terminal builder (match fdecl.styp with
479         A.Void -> L.build_ret_void
480       | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
481   in
482
483   List.iter build_function_body functions;
484   the_module
```

11.5 propeller.ml

```
1  (* Authors: Isra Ali, Gwendolyn Edgar, Randy Price, Chris Xiong *)
2  (* open Ast
3  open Sast *)
4
5  type action = Ast | Sast | LLVM_IR | Compile
6
7  let () =
8    let action = ref Compile in
9    let set_action a () =
10     action := a
11   in
12   let speclist = [
13     ("-a", Arg.Unit (set_action Ast), "Print the AST");
14     ("-s", Arg.Unit (set_action Sast), "Print the SAST");
15     ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
16     ("-c", Arg.Unit (set_action Compile), "Check and print the generated LLVM IR (default)")
       ;] in
17   let usage_msg = "usage: ./toplevel.native [-a|-s|-l] [file.pr]" in
18   let channel = ref stdin in
19   Arg.parse speclist (fun file -> channel:= open_in file) usage_msg;
20
21   let lexbuf = Lexing.from_channel !channel in
22   let ast = Parser.program Scanner.token lexbuf in
23   match !action with
24       Ast -> print_string (Ast.string_of_program ast)
25     | _ -> let sast = Semant.check ast in
26       match !action with
27           Ast -> ()
```

```
        | Sast    -> print_string (Sast.string_of_sprogram sast)
        | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
        | Compile ->
            let m = Codegen.translate sast in
            Llvm_analysis.assert_valid_module m;
            print_string (Llvm.string_of_llmodule m)

  (* let sast = Semant.check ast in
  print_string(Sast.string_of_sprogram sast) *)
```

11.6 sast.ml

```
(* Authors: Isra Ali, Gwendolyn Edgar, Randy Price, Chris Xiong *)
open Ast

type sobj_decl = {
  soname : string;
  sprops : bind list;
  sextern : bool }

type sexpr = typ * sx
and sx =
    SIliteral of int
  | SFliteral of float
  | SBliteral of bool
  | SSliteral of string
  | SLliteral of sexpr array
  | SCall of string * sexpr list
  | SAssign of string * sexpr
  | SSetprop of string * string * sexpr
  | SId of string
  | SGetprop of string * string
  | SIndex of string * sexpr
  | SBinop of sexpr * binop * sexpr
  | SUnop of unop * sexpr
  | SParentheses of sexpr
  | SNoexpr

type sstmt =
    SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt list * (sexpr * sstmt list) list * sstmt list
  | SFor of string * sexpr * sexpr * sstmt list
  | SWhile of sexpr * sstmt list
  | SBreak
  | SContinue
  | SBind of string * string * string
  | SUnbind of string * string * string

type sfunc_decl = {
  styp : typ;
  sfname : string;
  sformals : bind list;
  slocals : bind list;
  sbody : sstmt list }

type sprogram = bind list * obj_decl list * sfunc_decl list

let rec string_of_sexpr (t, e) = "(" ^ string_of_typ t ^ " : " ^ (match e with
    SIliteral x -> string_of_int x
  | SFliteral x -> string_of_float x
  | SBliteral x -> if x then "true" else "false"
  | SSliteral x -> x
```

```
52    | SLliteral xs -> "[" ^ String.concat ", " (Array.to_list (Array.map string_of_sexpr xs))
        ^ "]"
53    | SCall (f, es) -> f ^ "(" ^ String.concat ", " (List.map string_of_sexpr es) ^ ")"
54    | SAssign (id, e) -> id ^ " = " ^ string_of_sexpr e
55    | SSetprop (o, p, e) -> o ^ "." ^ p ^ " = " ^ string_of_sexpr e
56    | SId id -> id
57    | SGetprop (o, p) -> o ^ "." ^ p
58    | SIndex (id, e) -> id ^ "[" ^ string_of_sexpr e ^ "]"
59    | SBinop (e1, op, e2) -> string_of_sexpr e1 ^ " " ^ string_of_binop op ^ " " ^
        string_of_sexpr e2
60    | SUnop (op, e) -> (match op with
61        Not -> string_of_unop op ^ " (" ^ string_of_sexpr e ^ ")"
62      | Neg -> string_of_unop op ^ "(" ^ string_of_sexpr e ^ ")")
63    | SParentheses e -> "(" ^ string_of_sexpr e ^ ")"
64    | SNoexpr -> "") ^ ")"
65
66    let string_of_sodecl odecl =
67      if odecl.sextern then
68        "external objdef " ^ odecl.soname ^ "\n" ^
69        brace_wrap (String.concat "\n" (List.map string_of_vdecl odecl.sprops))
70      else
71        "objdef " ^ odecl.soname ^ "\n" ^
72        brace_wrap (String.concat "\n" (List.map string_of_vdecl odecl.sprops))
73
74  let rec string_of_sstmt = function
75      SExpr e -> string_of_sexpr e ^ ";"
76    | SReturn e -> (match e with
77          (Void, SNoexpr) -> "return;"
78        | _               -> "return " ^ string_of_sexpr e ^ ";")
79    | SIf (e, s1, elifs, s2) ->
80        let if_str =
81          "if " ^ string_of_sexpr e ^ "\n" ^
82          brace_wrap (String.concat "\n" (List.map string_of_sstmt s1)) in
83        let string_of_elif (elif_e, elif_s) =
84          "elif " ^ string_of_sexpr elif_e ^ "\n" ^
85          brace_wrap(String.concat "\n" (List.map string_of_sstmt elif_s))
86        in
87        let elif_str = match elifs with
88            [] -> ""
89          | _  -> "\n" ^
90                  String.concat "\n" (List.map string_of_elif elifs) in
91        let else_str = match s2 with
92            [] -> ""
93          | _  -> "\n" ^
94                  "else\n" ^
95                  brace_wrap(String.concat "\n" (List.map string_of_sstmt s2)) in
96        if_str ^ elif_str ^ else_str
97    | SFor (id, e1, e2, s) ->
98        "for " ^ id ^ " from " ^ string_of_sexpr e1 ^ " to " ^ string_of_sexpr e2 ^ "\n" ^
99        brace_wrap (String.concat "\n" (List.map string_of_sstmt s))
100   | SWhile (e, s) ->
101       "while " ^ string_of_sexpr e ^ "\n" ^
102       brace_wrap (String.concat "\n" (List.map string_of_sstmt s))
103   | SBreak -> "break;"
104   | SContinue -> "continue;"
105   | SBind (o, p, f) -> "bind(" ^ o ^ "." ^ p ^ ", " ^ f ^ ");"
106   | SUnbind(o, p, f) -> "unbind( " ^ o ^ "." ^ p ^", " ^ f ^ ");"
107
108   (* | _ -> "NONE" *)
109
110 let string_of_sfdecl fdecl =
111   "fn " ^ fdecl.sfname ^ "("  ^ String.concat ", " (List.map snd fdecl.sformals) ^ ") -> " ^
        string_of_typ fdecl.styp ^ "\n" ^
112   brace_wrap ((String.concat "\n" (List.rev (List.map string_of_vdecl fdecl.slocals))) ^ "\n
      \n" ^
```

```
              String.concat "\n" (List.map string_of_sstmt fdecl.sbody))

let string_of_sprogram (vdecls, odecls, fdecls) =
  String.concat "\n" (List.rev (List.map string_of_vdecl vdecls)) ^ "\n\n" ^
  String.concat "\n\n" (List.rev (List.map string_of_sodecl odecls)) ^ "\n\n" ^
  String.concat "\n\n" (List.rev (List.map string_of_sfdecl fdecls))
```

11.7 semant.ml

```
(* Authors: Isra Ali, Gwendolyn Edgar, Randy Price, Chris Xiong *)
open Ast
open Sast

module StringMap = Map.Make(String)

let check (globals, objects, functions) =

  (* global variables *)

  let check_binds kind to_check =
    let name_compare (_, n1) (_, n2) =
      compare n1 n2
    in
    let check_it checked binding = match binding with
        (Void, _) -> raise (Failure ("void " ^ kind))
      | (_, n1)   -> match checked with
          ((_, n2) :: _) when n1 = n2 -> raise (Failure ("duplicate " ^ kind))
        | _ -> binding :: checked
    in
    let _ = List.fold_left check_it [] (List.sort name_compare to_check) in
    to_check
  in

  let globals' = check_binds "global" globals in

  (* objects *)

  let check_obj odecl = {
    soname = odecl.oname;
    sprops = check_binds "property" odecl.props;
    sextern = odecl.extern }
  in

  let objects' = List.map check_obj objects in

  let add_obj map odecl = match odecl with
      _ when StringMap.mem odecl.soname map -> raise (Failure "duplicate objdef")
    | _ ->   StringMap.add odecl.soname odecl map
  in

  let object_decls = List.fold_left add_obj StringMap.empty objects' in

  let find_objdecl o =
    try StringMap.find o object_decls
    with Not_found -> raise (Failure ("undefined object " ^ o))
  in

  let get_prop p odecl =
    let f (_, name) =
      name = p
    in
    try  List.find f odecl.sprops
```

```
54      with Not_found -> raise (Failure ("object type " ^ odecl.soname ^ " has no property " ^
        p))
55    in
56
57    (* functions *)
58    let built_in_decls =
59      let add_bind map (name, t) = StringMap.add name
60        { typ = Void;
61          fname = name;
62          formals = [(t, "x")];
63          locals = [];
64          body = []; } map
65      in
66       List.fold_left add_bind StringMap.empty
67      [ ("print", Int); ("prints", Str); ("printf", Float); ("printb", Bool)] in
68
69    let add_func map fdecl = match fdecl with
70        _ when StringMap.mem fdecl.fname built_in_decls -> raise (Failure (fdecl.fname ^ " is
        already a built-in function"))
71      | _ when StringMap.mem fdecl.fname map            -> raise (Failure ("duplicate function
         " ^ fdecl.fname))
72      | _ -> StringMap.add fdecl.fname fdecl map
73    in
74
75    let function_decls = List.fold_left add_func built_in_decls functions in
76
77    let find_func f =
78      try StringMap.find f function_decls
79      with Not_found -> raise (Failure ("undefined function " ^ f))
80    in
81
82    let _ = find_func "init" in
83
84    let check_function func =
85      let loop_vars =
86        let rec find_loop_vars = function
87            [] -> []
88          | (st::sts) -> (match st with
89                            If (_, s1, elifs, s2) ->
90                              let elif_stmts = List.map snd elifs in
91                              (find_loop_vars s1) @ (List.flatten (List.map find_loop_vars
        elif_stmts)) @ (find_loop_vars s2) @ (find_loop_vars sts)
92                          | For (id, _, _, s) -> (Int, id) :: find_loop_vars s
93                          | While (_, s) -> (find_loop_vars s) @ (find_loop_vars sts)
94                          | _ -> find_loop_vars sts)
95        in
96        find_loop_vars func.body in
97
98      let check_locals kind to_check =
99        let name_compare (_, n1) (_, n2) =
100         compare n1 n2
101       in
102       let check_it checked binding = match binding with
103           (Void, _) -> raise (Failure ("void " ^ kind))
104         | (Custom o, n1) -> let _ = find_objdecl o in
105                             (match checked with
106                                 ((_, n2) :: _) when n1 = n2 -> raise (Failure ("duplicate "
        ^ kind))
107                               | _ -> binding :: checked)
108         | (_, n1)   -> match checked with
109             ((_, n2) :: _) when n1 = n2 -> raise (Failure ("duplicate " ^ kind))
110           | _ -> binding :: checked
111       in
112       let _ = List.fold_left check_it [] (List.sort name_compare to_check) in
113       to_check
```

```
    in

    let formals' = check_binds "formal" func.formals in
    let locals'  = check_locals "local" (func.locals @ loop_vars) in

    let check_assign lt rt msg =
      if   lt = rt
      then lt
      else raise (Failure msg)
    in

    let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
                                 StringMap.empty (globals' @ formals' @ locals') in

    let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undefined identifier " ^ s))
    in

    let rec expr = function
        Iliteral x -> (Int,   SIliteral x)
      | Fliteral x -> (Float, SFliteral x)
      | Bliteral x -> (Bool,  SBliteral x)
      | Sliteral x -> (Str,   SSliteral (Scanf.unescaped (String.sub x 1 ((String.length x)
    - 2))))
      | Lliteral xs ->
          let (ty, _) = expr (Array.get xs 0) in
          let eqty e =
            let (tt, _) = expr e in
            tt = ty
          in
          let tocheckedlit = function
              Iliteral e -> (Int, SIliteral e)
            | Fliteral e -> (Float, SFliteral e)
            | Bliteral e -> (Bool, SBliteral e)
            | Sliteral e -> (Str, SSliteral e)
            | _ -> raise (Failure "invalid literal in list")
          in
          let same = List.fold_left ( = ) true (List.map eqty (Array.to_list xs)) in
          if   not same
          then raise (Failure "unequal list element types")
          else (match ty with
                    Int   -> let sxs = Array.map tocheckedlit xs in
                             (List(Int), SLliteral sxs)
                  | Float -> let sxs = Array.map tocheckedlit xs in
                             (List(Float), SLliteral sxs)
                  | Bool  -> let sxs = Array.map tocheckedlit xs in
                             (List(Bool), SLliteral sxs)
                  | Str   -> let sxs = Array.map tocheckedlit xs in
                             (List(Str), SLliteral sxs)
                  | _     -> raise (Failure "bad list type"))
      | Id id -> (type_of_identifier id, SId id)
      | Getprop (o, p) ->
          let otype = type_of_identifier o in
          (match otype with
              Custom t ->
                let odecl = find_objdecl t in
                let (pty, _) = get_prop p odecl in
                (pty, SGetprop(o, p))
            | _        -> raise (Failure (o ^ " is not an object")))
      | Index (id, e) ->
          let ty = (match type_of_identifier id with
                      List(t) -> t
                    | _       -> raise (Failure ("cannot index non-list variable " ^ id)))
    in
```

```
177           let (t, e') = expr e in
178           (match t with
179               Int -> (ty, SIndex(id, (t, e')))
180             | _   -> raise (Failure "list index expression must of type int"))
181       | Call (f, es) ->
182           let fdecl = find_func f in
183           let fname = fdecl.fname in
184           let n_form = List.length fdecl.formals in
185           let n_args = List.length es in
186           if   n_args != n_form
187           then raise (Failure ("function " ^ fname ^ "expects " ^ (string_of_int n_form) ^
188                                " arguments, but was passed " ^ (string_of_int n_args)))
189           else
190           let check_call (ft, _) e =
191             let (et, e') = expr e in
192             let err_msg = "bad argument types in call to " ^ fname in
193             (check_assign ft et err_msg, e')
194           in
195           let es' = List.map2 check_call fdecl.formals es in
196           (fdecl.typ, SCall(f, es'))
197       | Assign (id, e) ->
198           let tid = type_of_identifier id
199           and (te, e') = expr e in
200           let err_msg = "illegal assignment to " ^ id in
201           (check_assign tid te err_msg, SAssign(id, (te, e')))
202       | Setprop (o, p, e) ->
203         let otype = type_of_identifier o in
204         (match otype with
205             Custom t ->
206               let odecl = find_objdecl t in
207               let (pt, _) = get_prop p odecl in
208               let (et, e') = expr e in
209               (check_assign pt et "illegal property assignment", SSetprop(o, p, (et, e')))
210           | _        -> raise (Failure (o ^ " is not an object")))
211       | Binop (e1, op, e2) ->
212           let (t1, e1') = expr e1
213           and (t2, e2') = expr e2 in
214           let ty = match op with
215               Add | Sub | Mlt | Div | Mod when t1 = t2 && t1 = Int   -> Int
216             | Add | Sub | Mlt | Div       when t1 = t2 && t1 = Float -> Float
217             | Eq  | Neq                   when t1 = t2              -> Bool
218             | Lt  | Leq | Gt  | Geq       when t1 = t2 && (t1 = Int || t2 = Float) -> Bool
219             | And | Or  | Xor             when t1 = t2 && t1 = Bool -> Bool
220             | _ -> raise (Failure "illegal binary operator") in
221           (ty, SBinop((t1, e1'), op, (t2, e2')))
222       | Unop (op, e) ->
223           let (t, e') = expr e in
224           let ty = match op with
225               Neg when t == Int || t == Float -> t
226             | Not when t == Bool -> Bool
227             | _ -> raise (Failure "illegal unary operator") in
228           (ty, SUnop(op, (t, e')))
229       | Parentheses e ->
230           let (ty, e') = expr e in
231           (ty, SParentheses (ty, e'))
232       | Noexpr -> (Void, SNoexpr)
233     in
234
235     let check_bool_expr e =
236       let (t', e') = expr e in
237       if   t' != Bool
238       then raise (Failure "expected boolean expression")
239       else (t', e')
240     in
241
```

```
242     let check_int_expr e =
243       let (t', e') = expr e in
244       if   t' != Int
245       then raise (Failure "expected int expression")
246       else (t', e')
247     in
248
249     let rec check_stmt in_loop = function
250         Expr e -> SExpr (expr e)
251       | Return e ->
252         let (ty, ex) = expr e in
253         let _ = check_assign ty func.typ ("bad return type in function " ^ func.fname) in
254         SReturn (ty, ex)
255       | If (e, s1, elifs, s2) ->
256           let check_elif (elif_e, elif_s) =
257             (check_bool_expr elif_e, List.map (check_stmt in_loop) elif_s)
258           in
259           let elifs' = match elifs with
260               [] -> []
261             | _  -> List.map check_elif elifs in
262           let s2' = match s2 with
263               [] -> []
264             | _  -> List.map (check_stmt in_loop) s2 in
265           SIf (check_bool_expr e, List.map (check_stmt in_loop) s1, elifs', s2')
266       | For (id, e1, e2, s) ->
267           SFor (id, check_int_expr e1, check_int_expr e2, List.map (check_stmt true) s)
268       | While (e, s) -> SWhile (check_bool_expr e, List.map (check_stmt true) s)
269       | Break    -> if not in_loop then raise (Failure "break outside loop") else SBreak
270       | Continue -> if not in_loop then raise (Failure "continue outside loop") else
    SContinue
271       | Bind (o, p, f) ->
272           let oty = type_of_identifier o in
273           (match oty with
274                Custom t ->
275                  let odecl = find_objdecl t in
276                  let (pty, _) = get_prop p odecl in
277                  let fdecl = find_func f in
278                  (match fdecl.formals with
279                      (fty1, _) :: (fty2, _) :: [] ->
280                         if fty1 != pty || fty2 != pty
281                         then raise (Failure "type of property must match type of bound
    function arguments")
282                         else
283                         SBind (o, p, f)
284                    | _ -> raise (Failure "bound functions must take 2 arguments"))
285              | _ -> raise (Failure (o ^ " is not an object")))
286       | Unbind (o, p, f) ->
287           let oty = type_of_identifier o in
288           (match oty with
289                Custom t ->
290                  let odecl = find_objdecl t in
291                  let _ = get_prop p odecl in
292                  let _ = find_func f in
293                  SUnbind (o, p, f)
294                | _ -> raise (Failure (o ^ " is not an object")))
295     in
296
297     { styp = func.typ;
298       sfname = func.fname;
299       sformals = formals';
300       slocals = locals';
301       sbody = List.map (check_stmt false) func.body }
302   in
303
304   let functions' = List.map check_function functions in
```

```
305   (globals', objects', functions')
```

11.8 default.c

```
1 extern int init();
2
3 int main() { return init(); }
```

12 Appendix E: Repository Directory Structure

```
Compilers
 |-documentation     Documents that are submitted in a deliverable,
 |  |                together with LaTeX source code
 |  |-lrm            Initial LRM submission
 |  |-report         Final report
 |-propeller         Compiler source code and wrapper script
 |  |-demos          Demo Propeller programs
 |  |-runtime        Source code for runtime environments
 |  |-tests          Test facility and test suites
 |-submissions       All archives submitted in a deliverable
```