ComPyled

A statically-checked adaptation of Python

Cameron Miller, Daniel Hanoch, Gabriel Clinger, George DiNicola

Agenda

1. ComPyled Overview - Daniel
2. Types, Operators - Daniel
3. Compiler Architecture - Gabriel
4. Syntax and Grammar - Gabriel
5. Language Features (implementation) - Cameron
6. Testing - George

Language Overview

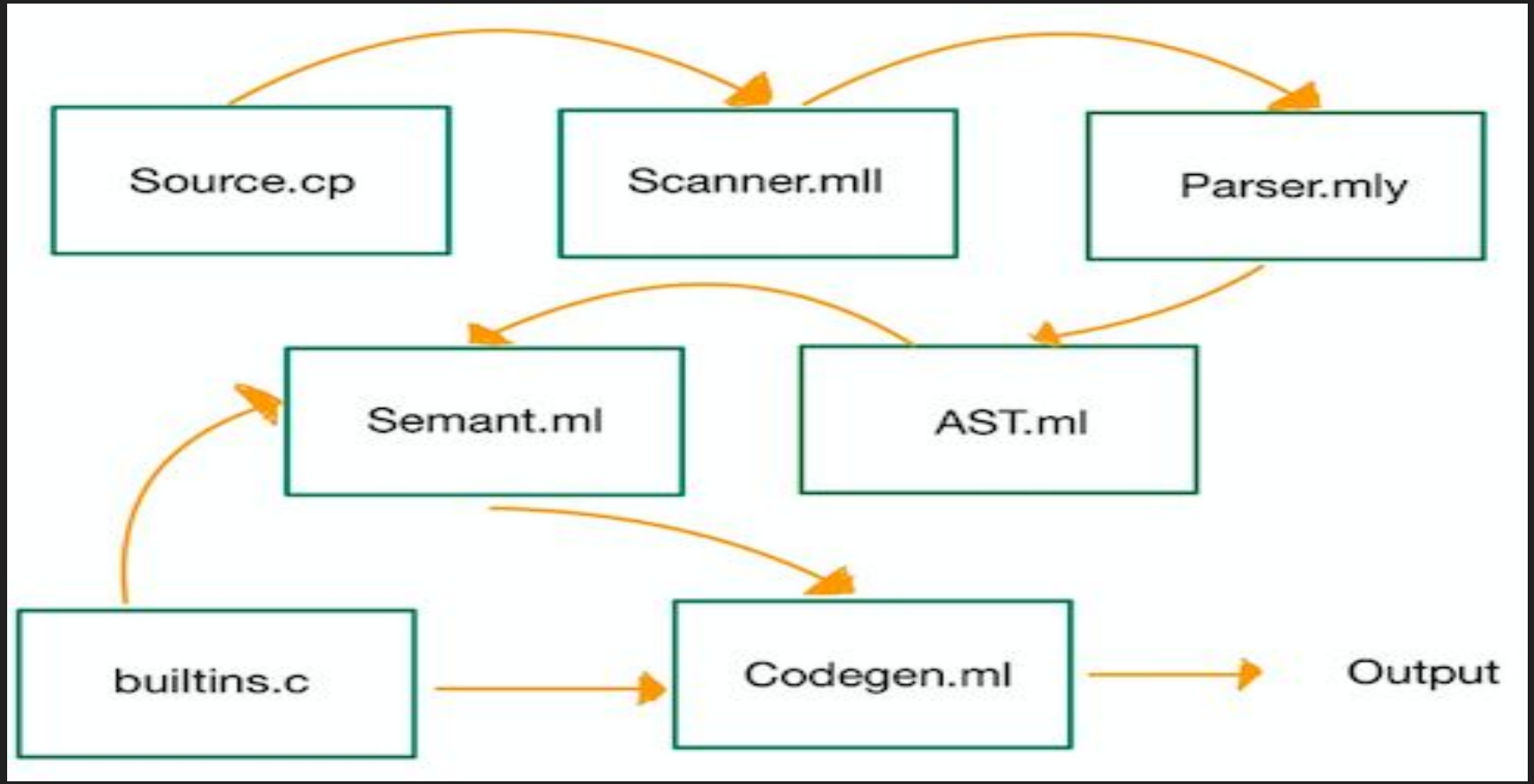
- Python + C = ComPyled
  - Statically-check language ≠ dynamically typed
  - Follows Python
  - C language - semicolon, curly braces

- Motivation:
  - Avoid indentation errors
  - Scoping ambiguities
  - User-friendly, general purpose language

Types, Operators, and Syntax

- Data Types:
  - Primitive: integer, float, boolean, string, void
  - Array data type
- Operators:
  - Assignment: =
  - Unary: not
  - Arithmetic: -, +, *, %, /
  - Comparison: <, >, ≤, ≥, ==, and, or
  - + operator is overloaded for string concatenation
- Built-in functions: len(object), print(object), overload of string concat (+) & string comparison (==)

Architecture Design



Syntax and Grammar

- Syntax - emulates a python style syntax mixed with C
- Brackets and semi-colons instead of indentation to signal expressions and end of statements
- Static typing instead of dynamic
- Types are determined at compile time

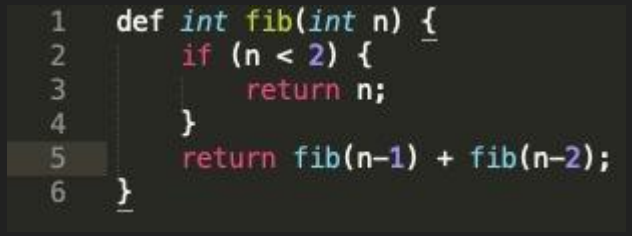
```
3   def void foo() {}
4
5   def int bar(int a, bool b, int c) {
6           if (b) {
7                   return c - a;
8           } else {
9                   return a + c;
10          }
11  }
12
13  print(bar(8, False, 10));
```

- No main but main is Ok
- Return or don’t return
- Assignment on declaration
- C for loops are just better

```
int i;
for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
}
```

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

}

def int main()
{
  int a;
  a = gcd(12, 18);

  print(a);
  return 0;
}

main();
```

- Recursion is a must
- Functions can be passed as arguments
- Strings are just arrays
- No memory management (no garbage collection either)



```
def int fib(int n) {
    if (n < 2) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}
```

Language Features

- “Hoisting” like javascript
- String Concatenation
- Arrays, len()
- Modulo

Language Features: Hoisting

```
1  print(x);
2
3  int x = 1;
```

Language Features: String Concatenation

```
string s1;
s1 = "hello ";

string s2;
s2 = " world!";

string s3;
s3 = s1 + s2;

print(s3);
```

```
hello world!
```

```
char *string_concat(const char *s1, const char *s2)
{
    char *result = malloc(strlen(s1) + strlen(s2) + 1);
    strcpy(result, s1);
    strcat(result, s2);
    return result;
}
```

Language Features: Arrays, len()

```
1  array x [int 3];
2  x[1] = 2.2;
3  print(x[1]);
```

Language Features: Modulo

```
1  print(3 % 3);
```

Testing

- Interesting Tests Examples

Interesting Test 1 (Hoisting Declarations and Initializations)

```
1
2  ## to ensure that evaluating x works regardless of
3  where it is declared/initialized ##
4
5  print(x);
6
7  int x = 1;
```

Outputs 1

```
1
2  ## to ensure that evaluating x works regardless of
3  where it is declared/initialized ##
4
5  print(x);
6
7  int x;
```

Outputs 0

Interesting Test 2: Declaring a Main Function

```
1   ## test for ensuring the user can still declare some main function ##
2
3   def int add(int x, int y)
4   {
5     return x + y;
6   }
7
8   def int main()
9   {
10    int a;
11    a = add(1, 3);
12
13    print(a);
14    return 0;
15  }
16
17  main();
```

Outputs 4

CodeGen Main Function Name “main0”

Interesting Test 3: Initializations inside of Functions

```
## purpose of this test is to initialize a local string var in a function ##

def string gen_str(int a) {
  string c = "hello";
  return c;
}

string s;
s = gen_str(3);
print(s);
```

Outputs Hello

```
## purpose of this test is to initialize a local var in a function ##

def int add_3_to(int a, int b) {
  int c = 3;
  return a + b + c;
}

int a;
a = add_3_to(39, 3);
print(a);
```

Outputs 45

Interesting Test 4: Initializing a Variable to Function Output

```
1  def int add(int a, int b) {
2      return a + b;
3  }
4
5
6  int x = add(1, 2);
7  print(x);
```

Outputs 3

Interesting Test 5: No Returns Outside of a Function

```
1  ## ensure user cannot return from a statement
2  outside of a function definition ##
3
4  int x;
5  x = 3;
6  return x;
```

Outputs

Fatal error: exception Failure("Return should not be specified without a function definition")

Interesting Test 6: Print Any Type Using “Print()”

- print(“hello world”);
  - Output: hello world
- print(1);
  - Output: 1
- int x = 1; print(x);
  - Output: 1
- print(True);
  - Output: 1
- print(4.2);
  - Output: 4.2

Interesting Test 7: if-else if-else - Nesting

```
## test that the if, else if, else can go to a sub-branch ##

int x;
int y;
y = 3;
x = 1;
if (x > 10) {
    print("went to if branch");
}
else if (x == 1) {

    if (y == 3) {
        print("went to the inner if in the elif branch");
    }
}
else if (x == 2) {
    print("went to second elif branch");
} else {
    print("went to else branch");
}
```

Outputs went to the inner if in the elif branch

Interesting Test 8: if-else if-else - Dangling Else

```
1   ## test that the dangling else does not occur ##
2
3   int x;
4   int y;
5   y = 3;
6   x = 1;
7   if (x > 10) {
8       print("went to if branch");
9   }
10  else if (x == 1) {
11
12      if (y == 2) {
13          print("went to the inner if in the elif branch");
14      }
15  }
16  else if (x == 2) {
17      print("went to second elif branch");
18  } else {
19      print("went to else branch");
20  }
```

Outputs Nothing

Future work

Garbage collection

String functions such as equality and splicing

Multi dimensional arrays

Thank you!