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

- Source.cp
- Scanner.mll
- Parser.mly
- Semant.ml
- AST.ml
- buildins.c
- Codegen.ml
- Output
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
- No main but main is Ok
- Return or don’t return
- Assignment on declaration
- C for loops are just better

```python
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)
Language Features

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

```plaintext
1       print(x);
2
3       int x = 1;
```
Language Features: String Concatenation

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

string s2;
s2 = " world!";

string s3;
s3 = s1 + s2;

print(s3);
```

```c
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()

```python
1    array x [int 3];
2    x[1] = 2.2;
3    print(x[1]);
```
Language Features: Modulo

```python
print(3 % 3);
```
Testing

- Interesting Tests Examples
Interesting Test 1 (Hoisting Declarations and Initializations)

```python
# to ensure that evaluating x works regardless of where it is declared/initialized 

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

Outputs 1

```python
# to ensure that evaluating x works regardless of where it is declared/initialized 

print(x);
int x;
```

Outputs 0
Interesting Test 2: Declaring a Main Function

```python
## test for ensuring the user can still declare some main function ##

def int add(int x, int y)
    { return x + y; }

def int main()
    {
        int a;
        a = add(1, 3);
        print(a);
        return 0;
    }

main();
```

Outputs 4
CodeGen Main Function Name “main0”
Interesting Test 3: Initializations inside of Functions

```python
## 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

```python
## 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

```python
def int add(int a, int b) {
    return a + b;
}

int x = add(1, 2);
print(x);
```

Outputs 3
Interesting Test 5: No Returns Outside of a Function

```cpp
// ensure user cannot return from a statement outside of a function definition

int x;
x = 3;
return x;
```

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Fatal error: exception Failure(&quot;Return should not be specified without a function definition&quot;)</th>
</tr>
</thead>
</table>
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

```python
## 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");

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

```
## test that the dangling else does not occur ##

```int` x;
`int` y;
y = 3;
x = 1;
if (x > 10) {
    print("went to if branch");
}
else if (x == 1) {
    if (y == 2) {
        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
Nothing
Future work

Garbage collection

String functions such as equality and splicing

Multi dimensional arrays
Thank you!