PCC Compiler

Yifei Yin, Litao (Jacob) Chen, Jathavan (Jat) Sellathurai

CSC488 | Group 12 | 2021 March

PCC (Python to C Compiler)

- A statically typed Python compiler
- Supports primitive types: integer, float, boolean
- Supports most of the arithmetic operations in Python
- Non-primitive types: list, tuple and string
- Function declaration, function call and loops

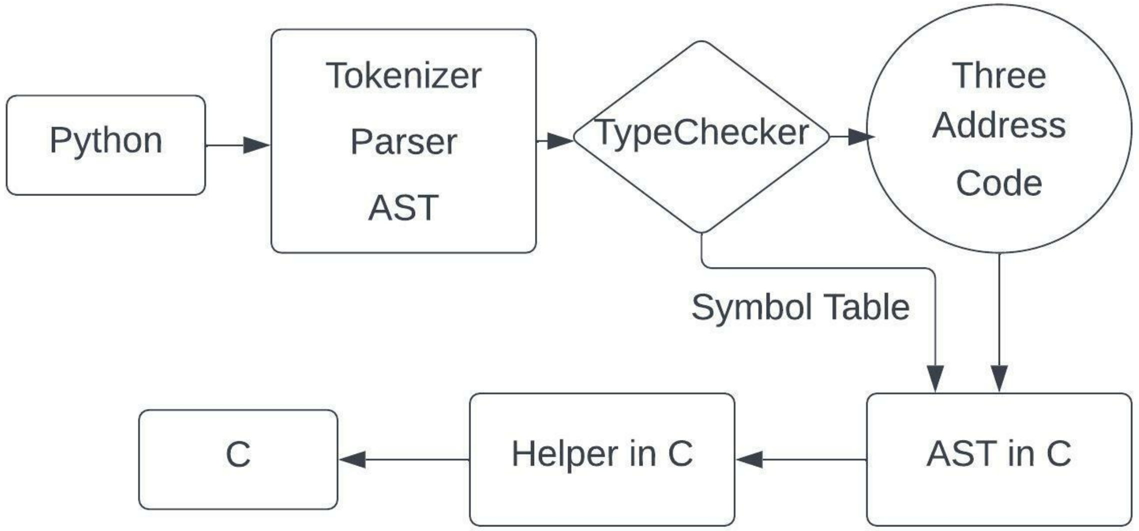
We Want To

- Build a Python Compiler
- Expand existing features in Python
- Enforce strict type checking in Python

Use Our Compiler

- If you want to pack your python program into executables for fast run speed and low cost.
- If you are interested in the performance analysis on interpreters and compilers.
- If you are a python lover or tired of programming in C.

Compiler Architecture



Primitive types & arithmetic operators

```python
positive_int: int = 10
negative_int: int = -10
zero_int: int = 0
positive_float: float = -10.0
negative_float: float = -10.0
zero_float: float = 0.0

a: int = 3 + 5
b: int = 4 * 7
c: int = 40 - a

b1: bool = True
b2: bool = False

sb: str = "Hello World!"
```

```c
int main() {
 /***** Main *****/
 int_t positive_int;
 positive_int = 10;
 int_t negative_int;
 negative_int = -10;
 int_t zero_int;
 zero_int = 0;
 float_t positive_float;
 positive_float = -10.0;
 float_t negative_float;
 negative_float = -10.0;
 float_t zero_float;
 zero_float = 0.0;
 int_t a;
 a = 8;
 int_t b;
 b = 28;
 int_t c;
 c = 40 - a;
 bool_t b1;
 b1 = true;
 bool_t b2;
 b2 = false;
 str_t sb;
 sb = "Hello World!";
 return 0;
}
```

Conditionals & loops

Input Code

```
a: int = 5
while a < 15:
   if a < 10:
       a  = a + 2
   else:
       print(a)
       a  = a + 1

b: int = 3
for i in range (b,10,2):
   print(i)
```

Target Code

```
int_t a;
a = 5;
while (a < 15) {
   if (a < 10) {
       a = a + 2;
   }
   else {
       print_int(a);
       a = a + 1;
   }
}
int_t b;
b = 3;
int_t i;
for (i = b; i < 10; i += 2){
   print_int(i);
}
```

Code Output

```
11
12
13
14
3
5
7
9
```

Functions

```
def foo(arg1: type, arg2:type ...) -> type:

    ...

    return type
```

Return type

Return Type Checking

```
def foo(x: int, y: int) -> int:
 a: float = 1.9
 return a
```

```
ParseError: Different type:
t1=Type(value=PrimitiveType(value='int'))
t2=Type(value=PrimitiveType(value='float'))
```

```
a: float = 1.9
b: bool = True

def foo(x: int, y: int) -> int:
   x = x + y
   return x

foo(a, b)
```

Undefined function?
I just defined it.
Why?

```
ParseError: ('Referencing undefined function "foo"')
```

Function Overloading

```python
def func(a: int) -> int:
   return a

def func(a: int, b: bool) -> int:
   return a
```

```
Natively Python does not support
function overloading. The upper
function will be overwritten by the
function at the bottom.
```

Our Goal

```python
def func(a: int) -> int:
  return a

def func(a: int, b: float) -> int:
  return a

d: int = func(2)
e: int = func(1, 1.9)
```

↓

Hashing!!!!

```c
int_t func2221(int_t a) {
   return a;
}
int_t func6381(int_t a, float_t b) {
   return a;
}
int_t d;
d = func2221(2);
int_t e;
e = func6381(1, 1.9);
```

```
d gets assigned 2
e gets assigned 1
```

Built-in IO

- `print`
- `input_`*`type`*
  - `input_int`
  - `input_float`
  - `input_bool`

```
input_int()

input_float(“pi = ?”)
```

Source:

- `print("hello world")`
- `print(42.0)`

Compiled:

- `print_str("hello world")`
- `print_float(42.0)`

```
Enter a number (expecting int): 1.0
Invalid input. Please try again.
Enter a number (expecting int): adasdas
Invalid input. Please try again.
Enter a number (expecting int):

pi = ? (expecting float):
```

```
hello world
42.0
```

Example Program

```python
n: int = input_int("How many number to add?")
result: int = 0
if n <= 0:
 print("You did not enter a positive number")

while n > 0:
 n = n - 1
 tmp: int = input_int()
 result = result + tmp

print("The sum is")
print(result)
```

```c
#include "../starter.c"

int main()
{
   /***** Main *****/
   int_t n;
   n = input_int_s("How many number to add?");
   int_t result;
   result = 0;
   if (n <= 0)
   {
       print_str("Yo
   }
   while (n > 0)
   {
       n = n - 1;
       int_t tmp;
       tmp = input_int();
       result = resu
   }
   print_str("The su
   print_int(result)

   /***** End of mai

   /***** Memory cle

   /***** End of Memory clean up       /

   return 0;
}
```

```
How many number to add? (expecting int): 3
Enter a number (expecting int): 1
Enter a number (expecting int): 2
Enter a number (expecting int): 3
The sum is
6
```

```
How many number to add? (expecting int): -1
You did not enter a positive number
The sum is
0
```

Extended types: list and tuple

```
List: [int], [float]

Tuple: (int), (float)
```

```
a: [int] = [1,2,3]
b: (int) = ()
c: (float) = (1.0, 2.0)

a.append(1)
print(a[0])
print(c[1])

c.append(1.0)
```

```
Exception: Cannot use append on type tuple
```

```
a: [int] = [1,2,3,4]
for elem in a:
   print(elem)
```

```
1
2
3
4
```

```
list_t * a = list_init(3);
list_init_add(int_v,a,1);
list_init_add(int_v,a,2);
list_init_add(int_v,a,3);

list_t * b = list_init(0);

list_t * c = list_init(2);
list_init_add(float_v,c,1.0);
list_init_add(float_v,c,2.0);

list_add(int_v,a,1);

print_int(list_get(int_v,a,0));
print_float(list_get(float_v,c,1));

list_free(a);
list_free(b);
list_free(c);
```

Optimization

Temp variable removal

```
int_t _t1;
_t1 = -10;
int_t ia;
ia = _t1;
int_t _t2;
_t2 = 0;
int_t ib;
ib = _t2;
```

```
int_t ia;
ia = -10;
int_t ib;
ib = 0;
```

Constant folding

```python
a: int = 3 + 5
b: int = 4 * 7
print(1 + 2 * 3)
```

```c
int_t a;
a = 8;
int_t b;
b = 28;
print_int(7);
```

Future improvements

- Optimization
    - Constant propagation
    - Dead code elimination
- Type Extension
    - Nested list
    - Expand string operations

That’s it. Thank you

Questions are welcome!