Thivagar Nadarajan
Haider Sajjad
Ahmad Al-Taha

Sprint 0 – Project Description

Project Overview

Our project aims to create a compiler which will read Python code (file format .py) and output C++ code (file format .cpp). The input is clearly defined and has the required elements to produce the core features we will need. The target is also defined because it will be in the form of an executable file which can be verified.

Challenges We'll Face

The main challenge we expect to fact involves scanning through white-space characters (spaces & tabs). In Python, the size of the initial indent of any line, determines the scope of that line. However, the subsequent white-space characters of that line can be ignored. This means we need to develop a strategy in handling white-space characters so that we don't lose the scope but also don't needlessly handle white-space characters that don't have any relevance.

Benefits of Our Compiler

C++ runs faster than Python code. However
Python is a simpler language to learn and use, whereas C++ has a greater learning curve. However, C++ runs much faster than Python. Creating this compiler will allow us to gain the usability benefits of Python while having the performance benefits of C++.

Indentation:

We plan to handle indentation similarly to Python's parser. Information on how indentation is parsed is given here: https://docs.python.org/3.3/reference/lexical_analysis.html#indentation. To paraphrase, indentation level (# of tabs before a line of code) of each line will be recorded with the use of a stack. With each new line parsed, we will check the number of tabs that precedes the line of code and compare this to the last line parsed. We will then create an indent or dedent token depending on whether the indentation level of the current line is higher or lower respectively.

Primitives

The following Python primitives will be supported: integers, strings, floats, and booleans. These are supported by C++ and we believe compiling them into C++ will be trivial.

Here is an example of translating primitive types:

```python
# Python
x = 5
y = True
z = 10.55
w = "Hello"
```

```cpp
// C++
int x = 5;
bool y = true;
double z = 10.55;
char* w = "Hello";
```

Thivagar Nadarajan
Haider Sajjad
Ahmad Al-Taha

Methods

We will be supporting Python function translations. This may prove challenging because Python isn't statically typed. In Python, the type of the return value and the type of the function arguments are inferred dynamically at run time. In C++ the types are set before run-time. This means that our compiler needs to infer the types of the return values and function arguments, so that they can be used in our resulting C++ translation. Here is an example of translating methods:

```python
# Python
def sum(x, y):
    return x + y
```

```cpp
// C++
int sum(int x, int y) {
    return x + y;
}
```

Conditionals

We will also be supporting "if" statements. Handling "if" statements will require attention to the following block of code that is indented after the initial "if' statement. We will compile "if' statements by first translating the statement itself into C++ (which involves translating the condition of the statement as well) then translating all code within the block. Braces will then surround the block of translated code to adhere to C++ syntax. Here is an example of translating "if" statements:

```python
# Python
if True:
    print("Hello")
else:
    print("World")
```

```cpp
// C++
if (true) {
    cout << "Hello";
} else {
    cout << "World";
}
```

Loops:

"for" and "while" loops will be supported by our compiler. Similar to "if" statements, we will compile these loops by first translating the loop statement ("for" statement or "while" statement) then translating the following block of code that is indented. Braces will then be placed around the compiled code to adhere to C++ syntax. Here is an example of translating loops:

```python
# Python
for i in range(5):
    print(i)
```

```cpp
// C++
for (int i = 0; i < 5; i++) {
    cout << i;
}
```

Thivagar Nadarajan
Haider Sajjad
Ahmad Al-Taha

Data structures:

Lists will be supported by our compiler. Our biggest obstacle to overcome is Python allows lists to contain different types at each index, which C++ does not allow. Our compiler must check the list to ensure all types match, or end compilation in an error otherwise. Here is an example of translating lists:

```python
# Python
lst = [0,1,2,3,4]
```

```cpp
// C++
vector <int> lst {0,1,2,3,4};
```

Final Result

Here is an example of a more complex example, which incorporates multiple components of our compiler:

```python
# Python
def foo():
    lst = [0,1,2,3,4]
    for i in range(5):
        print(lst[i])
```

```cpp
// C++
void foo() {
    vector <int> lst {0,1,2,3,4};
    for (int i = 0; i < 5; i++) {
        cout << lst[i];
    }
}
```