Types

Int

Since `long` has been removed in Python 3, we decide to give two options for integers when we are compiling to C. One type is `int32_t`, for which value is in $[-2^{31}+1, 2^{31}-1]$. The other type is `long long` if the value is in $[-2^{63}+1, 2^{63}-1]$. Any value out of `long long` bound will give an error.

- The equivalent output in C will be `int32_t` or `long long` type, according to the value.
- `int32_t` is compatible with `long long`.

Float

Python `float` is implemented by C `double`. We can just use direct translation. (`float` → `double`)

String

Strings can be translated from python to C by dynamically allocating memory for an char array with the length of the python string plus a null terminator. String operations such as concatenating strings can also be done by dynamically allocating memory for a new char array with the size of both strings, then writing the characters of both strings to the new char array.

Bool

Python boolean has two values `True` and `False`. We will map these two boolean values to the last bit of a byte. The value of `1` will stand for `True` and `0` for `False` (`bool` → `byte`).

None

We will likely use `null` in C to represent `None`.

List

List can be implemented using arrays in C. We will likely use a struct to store the pointer and length among other axillary information. This will result in some inefficiencies in certain cases and we can optimise it later.

1. A list with length n stores n elements all having the same type.
2. Lists must have non-negative size.

3. Lists are mutable, just like lists in python, meaning two variables could share the same underlying data (changes in one variable will reflect to the other).

Access Methods

```
.length() -> Int
.slice(start: Int) -> [T]
.slice(start: Int, end: Int) -> [T]
.at(index: Int) -> T
```

Modify Methods

```
.update(index: Int, newValue: T) -> None
.remove(index: Int) -> T
.remove(start: Int, end: Int) -> [T]
.insert(index: Int, newValue: T) -> None
```

Other Methods

```
.copy() ->[T]
```

Notes:
1. "`T`" refers to the value type of the list
2. The indices are 0 based.
3. If a method has list as return value, the returned list is a new array – the underlying data is not shared with any other variables
4. The indices in all of these methods can be any integer. Negative values indicate counting from the end of the list. i.e. `arr.at(-4)` will be equivalent to `arr.at(arr.length() - 4)`
5. If indices are out of bound, a run time error occurs.

Tuple

Tuples have the same methods as array, except it does not have any of the "modify methods".

Operators

Addition `+`, subtraction `-`, multiplication `*` and division `/` must have integer or float type as their operands. The type of the expression follows the priority of `float > long` `long > int32_t`. If any of the operands have higher priority, the type of expression will be the higher priority type.

Modulus `%`. As the float modulus in python produces inaccurate results, we will only allow integers on the right side of the modulus. The type of this expression is dominated by the type of identifier on the left hand side.

Operands for greater `(>, >=)` and less `(<, <=)` must have numerical types such as `int` or `float` in python and `int32_t` and `long long` in C. Without importing

additional libraries, C does not have a boolean type, but we can use bitwise operations to achieve the same computational goal for numerical values. In this case, greater and less than operators will return non-zero upon true or `0` upon false.

Statements

If-Block

If-block in python and C have similar structures. So it can be directly translated.

While-Block

While-block in python and C have similar structures. So it can be directly translated.

For-Loop

If the loop counter is in terms of range in python, it can be directly replaced by an integer counter in C. If the loop counter is an object, like a list, in python, we can use the length of the object as the counter and access the object in the first line of the body.

Function Declaration

In python it is not necessary to declare the return type of a function or the types for the parameter. However, since C requires the return type and parameter types to be part of the function definition, the compiler will enforce a rule that the python code will have types defined in the declaration. A python function with return type None will be treated the same as a void function in C. The function body can be translated like how the compiler will translate other pieces of python code, but will also have return statements matching when the python code has return statements.

Function Call

Function calls in python have similar structure to function calls in C. So it can be directly translated.

Example Programs

Program 1 Input

```
a: float = 10.0
b: int = input("Please enter a number: ")
print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a % b)
print(-a)
print(-b)
```

Program 1 Output

```
int main() {
    double a = 10.0;
    int_t b = input("Please enter a number: ");
    printf("%f", a + b);
    printf("%f", a - b);
    printf("%f", a * b);
    printf("%f", a / b);
    printf("%f", a % b);
    printf("%f", -a);
    printf("%f", -b);

    return 0;
}
```

Program 2 Input

```
a = []
a.insert(a.length(), 0)
a.insert(a.length(), 1)
a.insert(a.length(), fib(a.at(0), a.at(1)))
a.remove(0)
i: int = 100
while i > 0:
    i = i - 1
    a.insert(a.length(), fib(a.at(0), a.at(1)))
    a.remove(0)
    print(a.at(0))

def fib(x: int, y: int) -> int:
    return x + y
```

Program 2 Output

```
int_t fib(int_t, int_t);

int main() {

    list_t a = list_init();
    list_insert(list_length(a), 0);
    list_insert(list_length(a), 1);
    list_insert(list_length(a), list_length(a), (int_t)
fib((int_t) list_at(a, 0), (int_t) list_at(a, 1)));
    list_remove(a, 0);

    int_t i = 100;
    while (i > 0) {
        i = i - 1;
        list_insert(a, list_length(a), (int_t) fib((int_t)
list_at(a, 0), (int_t) list_at(a, 1)));
        list_remove(a, 0);
        printf("%d", (int_t) list_at(a, 0));
    }

    return 0;
}

int_t fib(int_t x, int_t y) {
    return x + y;
}
```