Programming Language Concepts

Programming Language Theory

Schedule

- This is Week 4: We have 3 lectures.
- Our original schedule: Everything on Wednesday.
- Next Wednesday (9/30) is a Holiday.
- So we will postpone Week 5 to 10/7.

Topics

- **Name and Binding**
- **Environments** and Blocks
- Scope Rules

Name

- What is a **Name**?

- Name is merely a sequence of characters to represent (or denote) another object.

- In most languages, names have a form of identifiers.

    - e.g.) alphanumeric tokens (v1, v2, func, etc.) or sometimes other symbols (+, -).

Name ≠ Object

- A ***name*** and an ***object*** it denotes ***are not the same thing***.

- One name can represent several different objects.

- Also, one object may have several different names.

Denotable Objects

- ***Denotable objects*** are the objects that we can give a name.

- Objects whose names are given ***by users***: variables, parameters, functions, user-defined types, etc.

- Objects whose names are given ***by PL***: primitive types, primitive operations, pre-defined constants.

Binding

- Association (or binding) between a name and an object it denotes can be created at various times.

- Although it is not theoretical, but practical, we use the terms “*static”* and “*dynamic”* for two principle phases.

- ***static***: Design of language, Program writing, Compile time

- ***dynamic***: Runtime

Language Design Bindings

- primitive types (int, double, etc), primitive operations (+, -, etc).

- Same thing can be denoted by different names in different languages.

  - Logical "AND" operations in Java and Python.

Java

```java
if(a > 0 && b > 0) {
    q = a / b;
}
```

Python

```python
if a > 0 and b > 0:
    q = a / b
```

Binding Times

- ***Program Writing***: programmers choose identifiers, which is a partial definition of bindings. Such bindings will be completed later.

- ***Compile Time***: while translating, a compiler allocates memory to some of the data structures, such as *global variables*.

- ***Runtime***: complete all bindings have not been created yet. *local variables, pointer variables*, etc.

Referencing Environment

- *Referencing Environment (or simply environment)*, is a set of bindings between names and objects exist at a specific point in the program at runtime.

- It is a set of bindings.

- For names and objects at a certain point of execution.

- Usually, we only refer to bindings not set up by language definition.

Declaration

- A ***Declaration*** is an introduction of a binding in the environment.

- int x;

- int func() {
  return 0;
  }

- public class Foo;

Various Cases

- Bindings between names and denotable objects.
    - Single Name - Different Objects
    - Single Object - Different Names
        - In Different Environments
        - In the Same Environment - *aliasing.*

Single Name Different Objects

- Here is an Example Java class.
- We have the same variable sum in two locations.
- Although their names are the same, they actually point to two different objects.

```
public class Example {
    public int sum; 1
    public int method() {
        int sum = 0;
            2
        return sum;
    }
}
```

Single Object Different Names

- In different environments, this is more common.
- Call by reference.
- Inside the method `put()`, a variable `list` denotes to the same `ArrayList`, `strings`.

```
public static void main() {
    List<String> strings = new ArrayList<>();
    put(strings, "Middle");
}

public static void put(List<String> list, String str) {
    list.set(list.size()/2, str);
}
```

environment 1

environment 2

This change will affect strings too.

How about Call by Value?

- Call by Value copies the value and passes it to a method.
- So it is a case of different objects with a single or different names.
- Inside the method `put()`, a variable `oldStr` denotes to a different object, with the same value as `str` in `main()`.

```
public static void main() {
    String str = "Before";
    put(str, "After");
}
public static void put(String oldStr, String newStr) {
    oldStr = newStr;
}
```

This change will **not** affect str.

Single Object Different Names

- In the same environments, this is more tricky.

- The case that a single object with different names is called ***aliasing***, and the different names are called ***aliases***.

- Consider the following C code snippet.

- What should be printed at the last line?

environment 1

```
Declare x, y →  int *x, *y;
Allocate heap memory →  x = (int *) malloc(sizeof(int));
* Dereference →  *x = 5;
y point to the same as x →  y = x;
                            *y = 10;
                            printf("%d\n", *x);
```

Summary

- Names and Denotable Objects
- Bindings and Binding Time
- Various cases for bindings between names and objects.