#### Programming Languages

Janyl Jumadinova

March 13, 2023

- A name is exactly what you think it is
  - Most names are identifiers
  - symbols (like '+') can also be names

- A name is exactly what you think it is
  - Most names are identifiers
  - symbols (like '+') can also be names
- A binding is an association between two things, such as a name and the thing it names.

- A name is exactly what you think it is
  - Most names are identifiers
  - symbols (like '+') can also be names
- A binding is an association between two things, such as a name and the thing it names.
- The **scope** of a binding is the part of the program (textually) in which the binding is active.

#### **Binding Time:**

the point at which a binding is created or, more generally, the point at which any implementation decision is made.

#### **Binding Time:**

the point at which a binding is created or, more generally, the point at which any implementation decision is made.

- language design time program structure, possible type
- language implementation time
  - I/O, arithmetic overflow, type equality (if unspecified in manual)

3 / 17

- program writing time
  - algorithms, names

- program writing time
  - algorithms, names
- compile time
  - plan for data layout

- program writing time
  - algorithms, names
- compile time
  - plan for data layout
- link time
  - -layout of whole program in memory

- program writing time
  - algorithms, names
- compile time
  - plan for data layout
- link time
  - -layout of whole program in memory
- load time

## More Implementation Decisions

- run time
  - value/variable bindings, sizes of strings
  - NOTE: run time includes
    - program start-up time
    - module entry time
    - elaboration time (point at which a declaration is first "seen")
    - procedure entry time

The terms **STATIC** and **DYNAMIC** are generally used to refer to things bound before run time and at run time, respectively.

- "static is a coarse term; so is "dynamic"

The terms **STATIC** and **DYNAMIC** are generally used to refer to things bound before run time and at run time, respectively.

- "static is a coarse term; so is "dynamic"

Binding Times are very important in programming languages!

- In general, early binding times are associated with greater efficiency
- Later binding times are associated with greater flexibility

7 / 17

- In general, early binding times are associated with greater efficiency
- Later binding times are associated with greater flexibility
- Compiled languages tend to have early binding times
- Interpreted languages tend to have later binding times

## Scope Rules - Control Bindings

- Fundamental to all programming languages is the ability to name data
  - i.e., to refer to data using symbolic identifiers rather than addresses

# Scope Rules - Control Bindings

- Fundamental to all programming languages is the ability to name data
  - i.e., to refer to data using symbolic identifiers rather than addresses
- Not all data is named! For example, dynamic storage in C or Pascal is referenced by pointers, not names

# Scope Rules - Control Bindings

- Fundamental to all programming languages is the ability to name data
  - i.e., to refer to data using symbolic identifiers rather than addresses
- Not all data is named! For example, dynamic storage in C or Pascal is referenced by pointers, not names

```
double *d = (double *)malloc(8);
*d = 3.14; /* No name is bound to the value 3.14 */
/* The name ''d'' is bound to the ADDRESS containing 3.14
*/
```

 The period of time from creation to destruction is called the LIFETIME of a binding.

9/17

- The period of time from creation to destruction is called the LIFETIME of a binding.
- If object outlives binding it's garbage.
- If binding outlives object it's a dangling reference.

- The period of time from creation to destruction is called the LIFETIME of a binding.
- If object outlives binding it's garbage.
- If binding outlives object it's a dangling reference.
- The textual region of the program in which the binding is active is its scope.

#### Storage Allocation mechanisms

- Static
- Stack
- Heap

#### Storage Allocation mechanisms

- Static
- Stack
- Heap

#### Static allocation for

- code
- globals
- static or own variables

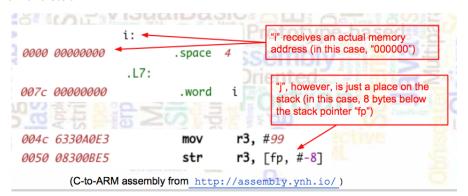
## Static Example

In C, variables can be global (visible to any function)

```
int i; /* i is global *
int f(int x) {
  return i+x;
                /* i is visible inside function f
main()
  int j;
                is visible only within main */
                is visible inside function main
      99;
               sets j to 13 */
```

## Static Example

When we compile this, i is stored in a fixed location, while j is allocated on the stack



12 / 17

## Two Types of Scoping

Static scoping (also called "lexical scoping")

- most familiar (Java, C)
- scope of variables known at compile time

# Two Types of Scoping

#### Static scoping (also called "lexical scoping")

- most familiar (Java, C)
- scope of variables known at compile time

#### **Dynamic scoping**

- scope depends on order of function calls at execution time
- pretty rare nowadays

# Static Scope Example (Java)

```
Scope of x,y
public static int x,y;
public static void main (String[] args) {
  x = 10; y = 20;
                                                                   Scope of x,y
  test();
                                                                   ("hole" in the
                                                                   scope of x,y)
    { int x = 70, y=80;
      System.out.println("x =
                                                                    Scope of W
  System.out.println("x = " + x + ", y = " + y);
                                                                    ("hole" in the
                                                                   scope of x,v)
public static void test() {
                                                                   OUTPUT:
    int = = 50, 7 = 60;
                                                                   x = 50, y = 60
    System.out.println("x = " + x + ", y = " + y);
                                                                   x = 70, v = 80
                                                                  x = 10, v = 20
```

# What Happens Here? (Java)

```
public static int x,y;
public static void main(String[] args)
  x = 10; y = 20;
  test1();
                                     Does this print "x = 10, y = 20"
public static void test1()
                                     or
                                     "x = 50, y = 60"?
    int x = 50, y = 60;
    test2();
public static void test2()
    System.out.println("x =
```

# What Happens in Dynamic Scoping?

```
int x,y;
start() {
                            Outputs "x = 50, y = 60"
  x = 10; y = 20;
  test1();
  test2();
                            Outputs "x = 10, y = 20"
test1()
    int x = 50, y = 60;
    test2();
test2()
    System.out.println("x =
```

#### In-Class Exercise

# JavaScript Case Study