Variables

A variable is a binding of a name to a memory address.

✦ Attributes of variables: type, value, lifetime, and scope.

✦ Binding a name to a memory address requires four pieces of information:

  • Name

  • Address: uniquely identifies the actual memory location

  • Type: even without explicit variable types, compiler or interpreter must internally handle typing

  • Value
l-value and r-value

\[ x = x + 1 \]

- Distinguish between a memory address and a value
- When a variable is placed at the left-hand side of an expression, the variable denotes an address.
- When a variable is placed at the right-hand side of an expression, the variable denotes a value.
- Since Algol68, the convention has been to call the memory address meaning of a variable a \textit{l-value}, and the value meaning of a variable as a \textit{r-value}. 
Value Manipulations of Variables in C

```c
int x;
int y;
int *z;

// l-value and r-value of x
x = x + 1;

// l-value of y, r-value of x
y = x + 1;

// l-value of z, l-value of x converted to an r-value
z = &x;

// l-value of x, dereferenced r-value of z
x = *z + 1;

// dereferenced l-value of z, r-value of y
*z = y;

// l-value of z, l-value of y converted to an r-value
z = &y + 4;

// l-value of x, l-value of z converted to an r-value
x = (int)z
```
Scope

The **scope** of a name is the **collection of statements** which can *access that name binding*.

✦ **Static scoping (lexical scoping)**

- A name is bound to a collection of statements in terms of its position in the source program (C, Java, Python)
- Good readability and better compile-time checking

✦ **Dynamic scoping**

- A name is bound to its most recent declaration in terms of the program’s execution history (Perl)
- Generally not used, since it makes type checking difficult and is prone to errors
Nested and Disjoint Scopes

✧ Disjoint scopes

- The same name can be bound to different entities without any interference (e.g. C function)

✧ Nested scopes

- Syntactic nesting: obvious in the layout of a program
- Semantic nesting: not obvious from the layout of a program (e.g. inheritance of Java, C++)
Example: Disjoint Scopes

```c
/**
 * File: power.c
 * Author: Ying Li
 * Date: 07/20/2015
 */

#include <stdio.h>

/* power: increase base to nth power; n >= 0 */
int power (int base, int n) {
    int i, power;
    power = 1;
    for (i = 1; i <= n; i++) {
        power = power * base;
    }
    return power;
}

/* test power function */
int main (int arg, char *argv[]) {
    int i;
    for (i = 0; i < 10; i++) {
        printf("%d %d %d \n", i, power(2, i), power(-3, i));
    }
    return 0;
}
```
Example: Syntactic Nesting

```java
/**
 * File: evenNum.c
 * Author: Ying Li
 * Date: 07/21/2015
 */

public class TimesTable {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            for (int j = 1; j <= 10; j++) {
                int result = i * j;
                System.out.printf("%3d", result);
            }
            System.out.println();
        }
    }
}
```
Example: Semantic Nesting

```
/**
 * File: Circle.java
 * Author: Ying Li
 * Date: 07/21/2015
 */

public class Circle {
    protected double radius;

    public Circle () {
        radius = 1.0;
    }

    public Circle (double r) {
        radius = r;
    }

    public double getArea () {
        return 3.14 * radius * radius;
    }

    public static void main (String[] args) {
        Circle c = new Circle();
        System.out.println("Area of the circle: " + c.getArea());
    }
}

/**
 * File: Cylinder
 * Author: Ying Li
 * Date: 07/21/2015
 */

public class Cylinder extends Circle {
    private double height;

    public Cylinder () {
        super();
        height = 1.0;
    }

    public Cylinder (double radius, double height) {
        super(radius);
        this.height = height;
    }

    public double getRadius () {
        return super.radius;
    }

    public double getVolume () {
        return getArea() * height;
    }

    public static void main (String[] args) {
        Cylinder c = new Cylinder(2.0, 1.0);
        System.out.println("Volume of the cylinder: " + c.getVolume());
        System.out.println("Radius of the cylinder: " + c.getRadius());
    }
}
```

Volume of the cylinder: 12.56
Radius of the cylinder: 2.0
Reference

**Defining Scope**: the scope in which a name is defined or declared.

**Non-local reference**: a reference to a name occurs in a nested scope of the defining scope.

**Local reference**: a reference which is not a non-local reference.
void timesTable (int bound) {
    int i, j;
    for (i = 1; i <= bound; i++) {
        for (j = 1; j <= bound; j++) {
            int result = i * j;
            printf("%3d ", result);
        } // end of for j
        printf("\n");
    } // end of for i
}

int main (int args, char *argv[]) {
    timesTable(9);
}
Example: Reference

```c
/**
 * File: timesTable.c
 * Author: Ying Li
 * Date: 07/21/2015
 */

void timesTable (int bound) {

    int i, j;

    for (i = 1; i <= bound; i++) {
        for (j = 1; j <= bound; j++) {
            int result = i * j;
            printf("%3d ", result);
        }
    }

    printf("\n");

}

int main (int args, char *argv[]) {
    timesTable(9);
}
```
Reference Rule

- **Forward reference**: a reference to a name can occur before the name has been declared (e.g. class fields and methods in java)

- **No forward reference**: a reference to a name can occur only after the name is declared (e.g. C)