Variables - I

SCS 2207: Programming Language Concepts

Kokila Perera

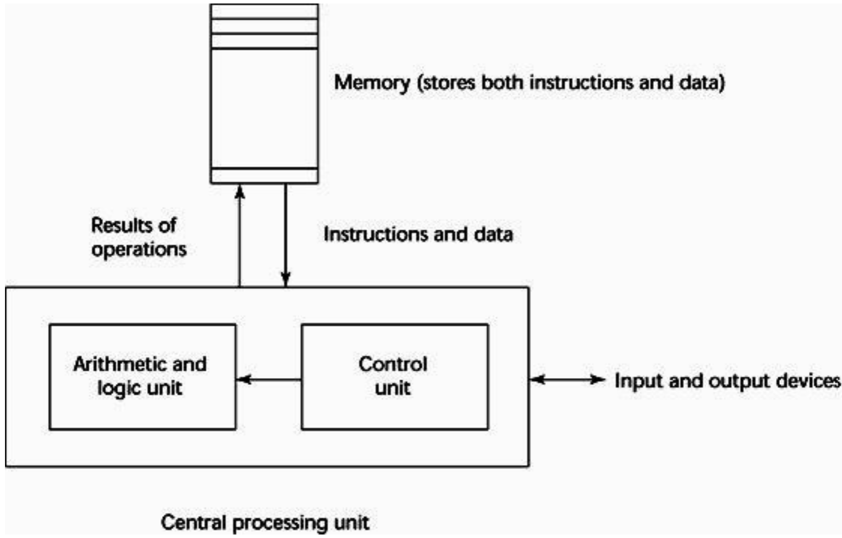
Outline

Introduction

Introduction

- Imperative languages : languages that are designed around von Neumann architecture

- Data and programs are stored in memory and separate from the CPU
  - **Variables model memory cells**
- Instructions and data are piped from memory to CPU
  - Assignment statements model piping
- The iterative form of repetition
  - Since instructions are stored in adjacent cells, iteration is more efficient than recursion



Introduction

Abstractions in a programming language for memory cells are variables.

The characteristics of the variables can be very close to the memory cells or can be far from the cells.

- Ex:
  - Integer variable - presented directly by one or more bytes of memory
  - 3 dimensional array - required a mapping function to support the abstraction

Why we need variables

Variables provide names for storage chunks to store data temporarily during the lifespan of a computer program.

```
x = y + z;
```

Variables are also used in programming to transfer information from one part of the program to another.

Eg : parameters, global variables

What are variables?

A variable is an identifier that is bind to a value stored in the system's memory (imperative languages) or an expression that can be evaluated (functional languages).

In imperative languages variables may be viewed as the abstraction in a language for a memory cells or a collection of cells.

Machine addresses are replaced by symbolic names.

Characteristics of variables

Characteristics of variables

Variables can be characterized by 6 attributes

- Name/ identifier
- Address (location/reference)
- Value
- Type
- Lifetime
- Scope

Characteristics of variables - Name

Variable names are the most common names used in programs.

Name is a label given to a memory cell(s)

Names

A fundamental attribute of variables.

Name is a string of characters that is used to identify entities in a program.

The term identifier is often used interchangeably with name.

Program Elements that may be Named

Names are also associated with subprograms, formal parameters, and other program constructs.

- **Variable names**
- Formal parameter names
- Subprogram names
- Names for data types
- Names for defined constants
- Statement labels
- Exception names
- Names for primitive operations (+,*, SQRT)
- Names for literal constants (17, 3.25)

Name Types

Consider the following two expressions.

```
a = a + 1

a[1] = a[1] + 1
```

What are **a** and **a[1]** ?

What are the differences between them?

Name Types

Simple Names

- Stands in its own without any modifiers to identify the entity

Composite Names:

- A name of a component of a data structure
- This would be simple name of the data structure + selection operation(s) for a particular component of the named structure

Name Types

Composite Names

- Examples 1: A[3]   (In C)
  - A is a simple name for the array
  - A[2] is a composite name

- Example 2: user.name   (In Java Scripts)
  - user : Object Name
  - name : element of that object.

Names

In compiled languages

- names only exists at compile time (unless compilation includes “debugging options”);
- at runtime only values and their memory addresses exist.

In interpreted languages

- names and values coexist during run-time.

Name Forms

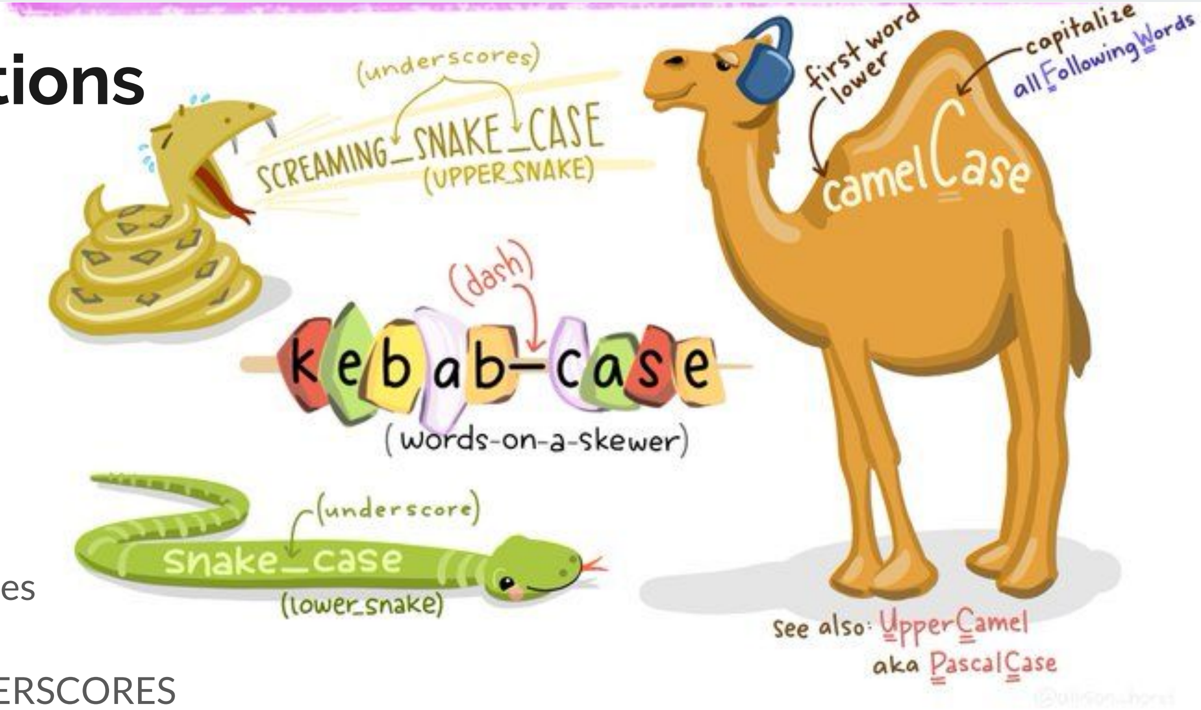
Names typically consists of

- **Letters, Digits, Connector** characters like underscore ('_')
    - Using connectors used to be a programming style choice (convention) in 1970's and 80's
    - Naming conventions used nowadays enforces using camel notation
    - Pascal, FOTRAN 77, Modula-2 do not allow any connector characters.

First characters

- Some languages have rules on the first character of the identifier.
    - In java, names should start with a letter
- In some languages special characters must be used.
    - Examples: PHP: $, PERL: $, @, %, RUBY: @ vs @@

Naming Conventions



- single lowercase letter
- single uppercase letter
- lowercase
- lower_case_with_underscores
- UPPERCASE
- UPPER_CASE_WITH_UNDERSCORES
- Pascal naming convention - capitalize the initial of each word.
- Camel naming convention - capitalize the initial of each word, except the first word.

Image source: https://twitter.com/allison_horst/status/1205702878544875521

Names - Length

Maximum length of a name

- Earliest programming languages used single character to name variables.
- Some languages impose restrictions on length of identifiers
    - In compilers a symbol table is used to store details about different program constructs
    - As identifiers are stored in symbol table while compiling which should not be too large length restrictions are imposed
    - Simplifies the maintenance of the symbol table

Names - Length

Maximum length of a name

- Length:
  - FORTRAN I : Maximum 6
  - COBOL : Maximum 30
  - FORTRAN 90 and ANSI C : Maximum 31
  - Ada : No limit, and all are significant*
    - There is no limit to the **length** of an **identifier** and the writer of the **compiler may impose a line length** limit.
    - (A line as a whole must adhere to the line length limit. Not the individual identifiers.)
  - C++ : no limit, but implementation often impose restriction.

Case Sensitivity

- Many languages are case sensitive. (ex: java and C-based languages)
- In case sensitive languages uppercase and lowercase letters in names are distinct

$$\left.\begin{array}{l}\circ\quad \text{name,}\\ \circ\quad \text{Name}\\ \circ\quad \text{NAME}\end{array}\right\}\ \text{3 different variables}$$

- Is case sensitivity of variables a good feature?

Case Sensitivity

- Disadvantages:
  - Poor readability: It violates the design principle that language constructs that look the same should have the same meaning
  - Poor writability: need to remember specific case usage to write correct program.
    - Ex: when using predefined methods in Java like parseInt
- Advantages
  - Larger namespace
  - Ability to use case sensitivity to group variables into different classes

Special Words

- Used to make programs more readable by naming actions to be performed
- Used to separate the syntactic parts of the statement and programs
- Special words are either keywords or reserved words

Special Words

- **Keyword**
- a word in programming language that is special only in certain contexts
  - Ex: In Fortran: Integer
  - If found at the beginning and followed by a word → a keyword (a declarative statement)
    - Integer count
  - If followed by an assignment operator → a variable name
    - Integer = 10

Special Words

- **Reserved Keywords**:
- Cannot be redefined by programmer (cannot be used as a name)
- As a language design choice ‘reserved’ keywords are better, as redefining can be confusing
    - Ex: in FORTRAN
        - Integer Real → declare an Integer type variable with name Real
        - Real Integer → declare a Real variable with name Integer
- If language has a large number of reserved keywords it may become difficult for programmer to making up names
    - Ex: COBOL has 300 reserved keywords
    - In languages like Java names are defined in another program units (packages) therefore visible only if explicitly imported. Once imported they cannot be redefined

Namespaces

A namespace is a context (container) for identifiers.

- Names in a namespace cannot have more than one meaning
- Names with different meanings cannot share the same name in the same namespace.
- Name spaces enable identifier names to be logically grouped.

Example : directory in OS.

Address

- Address of a variable is the machine memory address with which it is associated.
- Sometimes called as l-value of the variable.
    - Address is required on the left side when the name of a variable appears in an assignment

Address

- The same variable can be associated with ***different addresses at different times*** in the program
  - Ex: If a subprogram has a local variable that is allocated from runtime stack when the subprogram is called, different calls may result in that variable having different addresses.
  - These are in the sense different instantiation of the same variable
- The same variable can be associated with ***different addresses at different places*** in the program
  - The same variable name may be used in two subprograms of the same program

Address - Aliases

Aliases

- When more than one variable is used to access the same memory location they are called aliases
    - Ex: if total and sum are aliases, any change to total also apply to the sum and vice versa.
- Aliases are meant to save storage by allowing the data at a particular location to be used/viewed differently at different times

Aliases

Example : Consider the following two statements

```
x := a + b;

y := c + d;
```

This two statements look like two independent statements, thus interchanging their order may cause no problem for the final result.

What if x and c are aliases for the same data object ?

Aliases

Aliases

- Issues:
- Hinders readability - a variable value can be changed by assignment to a different variable
    - Therefore reader have to remember total and sum are different names for the same memory cell
- Makes program verification more difficult

Aliases

Aliases: Unions in C

Unions allow the same portion of memory to be accessed as different data types. All the names in the Union declaration refer the same memory location.

```
union union_name {
  member_type1 member_name1;
  member_type2 member_name2;
  member_type3 member_name3;
  .
} object_names;
```

All the elements of the union declaration occupy the same physical space in memory.
Its size is the one of the greatest element of the declaration

Aliases

Aliases: Unions in C

Example:

```
union mytypes_t {
        char c;
        int i;
        float f;
  } mytypes;
```

defines three elements:

mytypes.c

mytypes.i

mytypes.f

Each variable in the union has a different data type.

Since all of them are referring to the same location in memory, the modification of one of the elements will affect the value of all of them.

We cannot store different values in them independent of each other.

Aliases

Different languages provide different ways to create aliases

- FORTRAN EQUIVALENCE
- Pasca, Ada - Variant record structures
- Pointers
- Reference variables
- C and C++ Unions

Type

The type of a variable determines:

- The **range of values** the variable can store
- The **set of operations** that are defined for the values of the type

Example: int type in java

Range of values: -2147483648 to 2147483647

Set of operations: arithmetic operations for addition, subtraction, multiplication, division and modulus

Value

The value of a variable is the contents of the memory cell or cells associated with the variable

Sometimes called as r-value of the variable

- Because when the name appears in an assignment statement values is required on right side
- To access r-value, l-value must be determined first.

Value

It is convenient to think of computer memory as in terms of abstract cells rather than physical cells.

An abstract memory cell has the size required by the variable with which its associated.

- Ex:
- A floating point value may occupy four physical memory cells (assuming each one is byte-size)
- A floating point value is thought of as occupying a single abstract memory cell

Value

Variable interpolation (variable substitution/ variable expansion)

- The process of evaluating an expression or string literal containing one or more variables, yielding a result in which the variables are replaced with their corresponding values in memory.

  Example :

  ```
  $name = “someone”;

  print “Hello $name”;
  ```

  This feature is supported by the languages such as PHP, Perl, Ruby.

Value

Variable interpolation

Should the variable interpolation happen in all occurrences of variables in expressions?

- In PHP variable interpolation happen only in expression with double quotations.

Ex: Ruby:

```
name = "Ada"
puts "Hello, #{name}!"
```

PHP

```
$name = "Ada";
echo "Hello, $name":
```

Value

Ex: X =12

X names a place where the value 12 is now stored. What does the statement X = X + 1 mean ?

X = X + 1

X is used in two ways

- Address of the variable : l-value
- Value of the variable : r-value

Value

Dereferencing

Obtaining a value of a variable

In some programming all names stand for their references

Special form is required for dereferencing.

Ex: x := .x + 1

Value

Dereferencing

```
const char* p = "abc";
```

```
Memory Address (hex)    Variable name    Contents
1000                                     'a' == 97 (ASCII)
1001                                     'b' == 98
1002                                     'c' == 99
1003                                     0
...
2000-2003               p                1000 hex
```