Names, Bindings, and Scopes

Lecture 08

Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu

Introduction

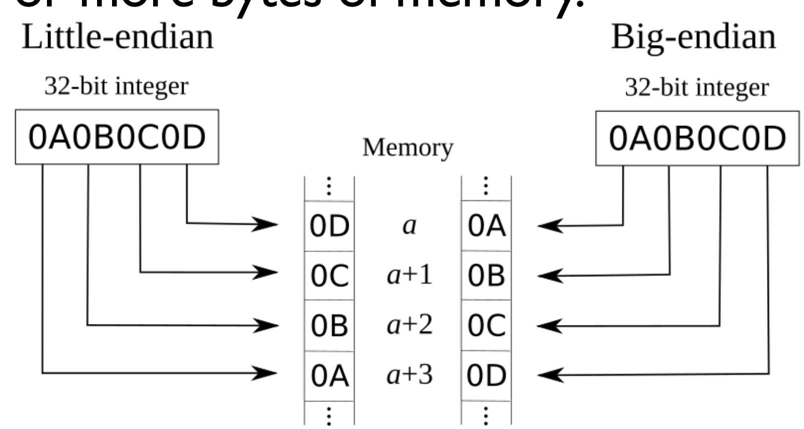
- Imperative programming languages are abstractions of the underlying Von Neumann computer architecture
  - Imperative programming languages:
    - Use statements to change program's state
    - Run statements one by one
  - Von Neumann architecture
    - Central Processing Unit (CPU)
    - Memory (stores data and instructions)
    - Input and output mechanism
    - External storage
- The architecture's two primary components
  - Memory, which stores both instructions and data
  - Processor, which provides operations for modifying the contents of the memory

Introduction

- The abstractions in a language for the ***memory cells*** of the machine are ***variables***

- In some cases, the characteristics of the abstractions are very close to the characteristics of the cells
  - An example of this is an integer variable, which is usually represented directly in one or more bytes of memory.

```
 0  :  00000001 00000000 00000000 00000000  |  1
 4  :  00000010 00000000 00000000 00000000  |  2
 8  :  00000011 00000000 00000000 00000000  |  3
12  :  00000100 00000000 00000000 00000000  |  4
16  :  00000101 00000000 00000000 00000000  |  5
```



The order of the bytes is called the **endianness**; left to right is **little endian**, because the least significant byte, the byte representing the smallest part of the number, comes first.

Introduction

- A variable can be characterized by a collection of ***properties***, or ***attributes***, the most important of which is ***type***, a fundamental concept in programming languages.

- Designing the data types of a language requires that a variety of issues be considered.

- Among the most important of these issues are the ***scope*** and ***lifetime*** of variables.

Names

- Before beginning our discussion of variables, the design of one of the fundamental attributes of variables, ***names***, must be covered.
- Names are also associated with subprograms, formal parameters, and other program constructs.
- The term ***identifier*** is often used interchangeably with name.

Design Issues

- The following are the primary design issues for names:
  - Are names ***case sensitive***?
  - Are the ***special words*** of the language ***reserved words*** or ***keywords***?

Name Forms

- A ***name*** is a string of characters used to identify some entity in a program.
- Fortran 95+ allows up to 31 characters in its names.
  - It has no more than 31 characters
  - The first character must be a letter,
  - The remaining characters, if any, may be letters, digits, or underscores,
  - Fortran identifiers are ***case insensitive***. That is, Smith, smith, sMiTh, SMiTH, smitH are all identical identifiers.
  - Correct Examples:
    - MTU, MI, John, Count
    - I, X
  - Incorrect Examples:
    - M.T.U.: only letters, digits, and underscores can be used
    - R2-D2: same as above

Name Forms

- A ***name*** is a string of characters used to identify some entity in a program.

- C99 has no length limitation on its internal names, but only the first 63 are *significant*.
- *External names* in C99 (those defined outside functions, which must be handled by the linker) are restricted to 31 characters.

- Names in Java, C#, and Ada have no length limit, and all characters in them are significant.
- C++ does not specify a length limit on names, although implementers sometimes do.

Name Forms

- Names in most programming languages have the same form:
  - a letter followed by a string consisting of letters, digits, and underscore characters (_).

- In the C-based languages, it has to a large extent been replaced by the so-called camel notation
  - All of the words of a multiple-word name except the first are capitalized, as in myStack
  - Language-specific conventions, https://en.wikipedia.org/wiki/Naming_convention_(programming)

- Note that the use of underscores and mixed case in names is a ***programming style*** issue, not a language design issue.

Name Forms

- All variable names in PHP must begin with a ***dollar sign***.

```php
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

Name Forms

- In Perl, the special character at the beginning of a variable's name, $, @, or %, specifies its type
  - $, a scalar value
  - @, an array
  - %, key/value pair

```
$age = 25;              # An integer assignment
$name = "John Paul";    # A string
$salary = 1445.50;      # A floating point

print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

Name Forms

- In Perl, the special character at the beginning of a variable's name, $, @, or %, specifies its type
  - $, a scalar value
  - @, an array
  - %, key/value pair

```
@ages = (25, 30, 40);
@names = ("John Paul", "Lisa", "Kumar");

print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

Name Forms

- In Perl, the special character at the beginning of a variable's name, $, @, or %, specifies its type
  - $, a scalar value
  - @, an array
  - %, key/value pair

```
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);

print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

Name Forms

- In Ruby, special characters at the beginning of a variable's name, @ or @@, indicate that the variable is an instance or a class variable, respectively.

Name Forms

- In many languages, notably the C-based languages, ***uppercase*** and ***lowercase*** letters in names are ***distinct***; that is, names in these languages are ***case sensitive***.
  - For example, the following three names are distinct in C++: rose, ROSE, and Rose.

- To some people, this is a serious detriment to readability, because names that look very similar in fact denote different entities.
  - In that sense, case sensitivity violates the design principle that language constructs that look similar should have similar meanings.
  - But in languages whose variable names are case-sensitive, although Rose and rose look similar, there is no connection between them.

Name Forms

- In C, the problems of case sensitivity are avoided by the convention that variable names do not include uppercase letters.
  - C library guide:
    - http://www.fortran-2000.com/ArnaudRecipes/Cstd/

- In Java and C#, however, the problem cannot be escaped because many of the predefined names include both uppercase and lowercase letters.
  - For example, the Java method for converting a string to an integer value is parseInt, and spellings such as ParseInt and parseint are not recognized.

Special Words

- ***Special words*** in programming languages are used to make programs more readable by naming actions to be performed.
- They also are used to separate the syntactic parts of statements and programs.
- In most languages, ***special words*** are classified as ***reserved words***, which means they cannot be redefined by programmers, but in some they are only ***keywords***, which means they can be redefined.

Special Words

- A ***keyword*** is a word of a programming language that is special only in certain contexts.
- Fortran is the only remaining widely used language whose special words are keywords.
- In Fortran, the word ***Integer***, when found at the beginning of a statement and followed by a name, is considered a keyword that indicates the statement is a declarative statement.
- However, if the word ***Integer*** is followed by the assignment operator, it is considered a variable name.

```
Integer Apple
Integer = 4
```

- Fortran compilers and people reading Fortran programs must distinguish between names and special words by context.

Special Words

- A ***reserved word*** is a special word of a programming language that cannot be used as a name.
- As a language design choice, reserved words are better than keywords because the ability to redefine keywords can be confusing.
- For example, in Fortran, one could have the following statements:

```
Integer Real
Real Integer
```

  - These statements declare the program variable Real to be of Integer type and the variable Integer to be of Real type.
  - In addition to the strange appearance of these declaration statements, the appearance of Real and Integer as variable names elsewhere in the program could be misleading to program readers.

Variables

- A program ***variable*** is an abstraction of a computer memory cell or collection of cells.
- Programmers often think of variable names as names for memory locations, but there is much more to a variable than just a name.
- A variable can be characterized as a sextuple of attributes: (*name*, *address*, *value*, *type*, *lifetime*, and *scope*).
- Although this may seem too complicated for such an apparently simple concept, it provides the clearest way to explain the various aspects of variables.

Variables: Address

- The ***address*** of a variable is the ***machine memory address*** with which it is associated.

- This association is not as simple as it may at first appear.

- In many languages, it is possible for the same variable to be associated with different addresses at different times in the program.

- For example, if a subprogram has a local variable that is allocated from the run-time stack when the subprogram is called, different calls may result in that variable having different addresses.
  - These are in a sense different instantiations of the same variable.

Variables: Address

- The address of a variable is sometimes called its ***l-value***, because the address is what is required when the name of a variable appears in the left side of an assignment.
- It is possible to have multiple variables that have the same address.
- When more than one variable name can be used to access the same memory location, the variables are called ***aliases***.
- Aliasing is a hindrance to readability because it allows a variable to have its value changed by an assignment to a different variable.
  - For example, if variables named total and sum are aliases, any change to the value of total also changes the value of sum and vice versa.
- Aliasing also makes program verification more difficult.

Variables: Type

- The type of a variable determines the range of values the variable can store and the set of operations that are defined for values of the type.
- For example, the int type in Java specifies a value range of -2147483648 to 2147483647 and arithmetic operations for addition, subtraction, multiplication, division, and modulus.

Variables: Value

- The value of a variable is the contents of the memory cell or cells associated with the variable.
- It is convenient to think of computer memory in terms of abstract cells, rather than physical cells.
- The physical cells, or individually addressable units, of most contemporary computer memories are byte-size, with a byte usually being eight bits in length.
- This size is too small for most program variables. An abstract memory cell has the size required by the variable with which it is associated.
- A variable’s value is sometimes called its ***r-value*** because it is what is required when the name of the variable appears in the right side of an assignment statement.