Fundamental Programming Principles: Variables and Data Types

Beyond the Mouse
GEOS 436/636
Jeff Freymueller, Sep 5, 2017

“You’ll never find a programming language that frees you from the burden of clarifying your ideas.”

“The Uncomfortable Truths Well”,
http://xkcd.com/568 (April 13, 2009)
Today’s Schedule

• How does computer programming work
  – What is a programming language?
  – What is a program?

• Variables and Data Types
  – How do we store values of different kinds?
    • Numbers
    • Strings of text
    • More complicated things (like images, for example)
Definitions

• A **programming language** is an *unambiguous* artificial language that is made up of a set of symbols (vocabulary) and grammatical rules (syntax) to instruct a machine.

• A **program** is a set of instructions in one or multiple programming languages that specifies the behavior of a machine.

• **Compilation** or **interpretation** is the verification of a program and its translation into machine readable instructions of a specific platform.
What Language Does the CPU Understand?

• The CPU (Central Processing Unit) actually understands only a language composed entirely of numbers, like this:
  – “157 65530 22 77 854” (this is a made-up example)
  – This means “execute instruction #157 using an argument 65530, then execute instruction #22, then execute instruction #77 using an argument 854”
  – The language definition tells the machine that instruction #157 takes one argument, but #22 does not.

• It is possible for a person to write code in this machine language, but almost nobody does it any more because it is so inconvenient.
I Actually Did This

PROGRAM: 6502 DISASSEMBLER

BY: JEFF FREYMUELLER

6502 ASSEMBLY LANGUAGE

!* = $7000

INIT

LDA #<START

LDAX START

STA MLMVGE C (03FA)

STA MLMVGC+1

RTS

START

CMP #D

BER OVERI

ERROR (E7F7)

ERRORPR

JMP OVERI

OVERI

JSR RDOB (E796) ; SKIP A SPACE

JSR RDOA (E7A7) ; READ ADDRESS

BCC ERRORPR ; IF CARRY = 0, ERROR

JSR TMPS (E797) ; SWAP TMPD & TMP2
Programming Languages

• Can be broken into two large families:

  • **Interpreted** languages. An interpreter program takes in commands, check syntax and translates to machine language at runtime (e.g., Matlab, Unix Shell)

  • **Compiled** languages. Programs are translated and saved in machine language by a compiler. At runtime no additional interpretation is necessary (e.g., FORTRAN, C/C++).
    
    — These generally run much faster than interpreted languages
Now, How Does Programming Work?

1. Open a text editor (MATLAB editor, vi, notepad, Text Wrangler, ... not MS Word)

2. translate your (physical or mental) flowchart into a set of instructions according to the rules of a programming language

3. test your program for syntactical correctness (ask the interpreter/compiler)

4. if errors, fix them and go back to (3)

5. test your program for semantic errors (the “fun” part!)

6. if errors, fix them and go back to (3)
Example: Hello World

1  >> dsp(halo orld
2    ??? dsp(halo orld
3        Error: Unexpected MATLAB expression.
4  >> dsp('halo_orld
5  ???_dsp('halo_orld
6        Error: A MATLAB string constant is not terminated properly.
7  >> dsp('halo_orld ')
8    ??? dsp('halo_orld ')
9        Error: Expression or statement is incorrect—possibly unbalanced (, {, or [..
10  >> dsp('halo_orld ')
11  ??? Undefined function or method 'dsp' for input arguments of type 'char'.
12  >> disp('halo_orld ')
13  halo orld
14  % Sematically correct, if you want to say 'hi' to the world:
15  %
16  >> disp('hello_world ')
17  hello world
The MATLAB Editor Helps You

disp('Hello world')

z = linspace(1,10,10);
s = sinh(z) + cosh(z);

foo = bar + ;
The MATLAB Editor Helps You
The MATLAB Editor Helps You

disp('Hello world')
z = linspace(1,10,10);
s = sinh(z) + cosh(z);
foo = bar + ;
What is a Variable?

- **Donald Knuth**: A quantity that may possess different values as a program is being executed.
- **Mehran Sahami**: A box in which we stuff things – i.e. a box with variable content.
- **Wikipedia**: User defined keyword that is linked to a value stored in computer’s memory (runtime).
- The concept of a variable consists of:
  - Name
  - Type
  - Value
Variables: Name

• USE MEANINGFUL NAMES!
• Must follow programming language rules
  – MATLAB variable names must begin with a letter, followed by any combination of letters, digits, and underscores. MATLAB distinguishes between uppercase and lowercase. No reserved keywords!
• USE MEANINGFUL NAMES, i.e. names that speak:
  ‘lengthGlacier’ or ‘glacier_length’ NOT NOT NOT ‘a’ – avoid ambiguity
• use consistent formatting, i.e.: ‘my_cool_var’ or ‘myCoolVar’ – this is easier to read
• a gazillion style guides exist – punchline: use meaningful names, be consistent (that’s hard enough)!
Variables: Type

• What is a type? – Think of sets of numbers in math: N,R,Z, ... The type refers to how numbers are being represented in a computer’s memory, i.e. which bit has which meaning, and how many bits are necessary

• primitive, built in types – for MATLAB e.g.: ‘int32’, ‘double’, ‘boolean’ (important for *printf functions)

• complex, home made types – (arrays,) structs, cell arrays (Matlab), classes
Variables: Type and Type Conversion

• some languages, e.g. MATLAB, shells, Perl are *weakly typed*: they do automatic type conversions (one type can be treated as another)
  – this is nice at first, occasionally this leads to nasty/hard to find problems (e.g. string interpreted as number, etc.)

• Other languages are very picky and will tell you that you can’t add a real number to a complex number without explicitly converting.
  – Why? It can produce more efficient machine code.
  – Picky vs loose is a design decision
Variables: Value

• A value of the type of the variable: 42, 3.1415926..., false, ‘text string’, i.e., the thing we stuff in the box

• Values can/should change during the runtime of the program. Some languages (not MATLAB) allow you to define a named \textbf{constant}, for values that can’t change.

• We need to be able to assign values to variables, and also access (dereference) the values.
Assignment and Access

• Assignment: set the value of a variable
  – MATLAB: `num_glaciers = 105`
  – tcsh scripting: `set filename = “12jun30dena.dat”`

• Access: get the value of a variable
  – MATLAB: `disp( num2str(num_glaciers) )`
  – tcsh scripting: `echo $filename`

• What does this do? (MATLAB)
  – `num_glaciers = num_glaciers + 1`
MATLAB Treats Everything as a Matrix

- Arrays or matrices are lists, vectors, matrices of data (1 to n dimensional)
- Therefore instead of one value they hold a list of values linked to a chunk of memory (a sequence of boxes)
- Access by index number: `glaciers(5), cov(3,2)`
- Shells allow only vectors (1-D arrays).
Example Arrays

• A numeric array:

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0</td>
<td>-3.2</td>
<td>1000</td>
<td>NaN</td>
<td>1</td>
<td>5</td>
<td>-90</td>
<td>9999</td>
<td>3.141</td>
<td>0</td>
</tr>
</tbody>
</table>

• Values can be a mix of integers, real and complex numbers.

  >> foo = [1; 2; 3+i; 4]

  foo =
  1.0000
  2.0000
  3.0000 + 1.0000i
  4.0000

• You can browse these values in the variable browser within the MATLAB GUI.
Example Arrays

- A string array:

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>H</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
<td></td>
<td>W</td>
<td>o</td>
<td>r</td>
<td>k</td>
</tr>
</tbody>
</table>

- Examples of assigning and accessing strings:
  
  ```
  >> foo = 'Hello Work'
  >> foo(4)
  ans =
    'l'
  >> foo(1)
  ans =
    'H'
  >> foo(1) + 1
  ans =
    73
  ```

What is going on here!?
### Setting up a numeric Matrix: Equinox marathon pacing tables

<table>
<thead>
<tr>
<th>index</th>
<th>Mile</th>
<th>record</th>
<th>well trained</th>
<th>mildly trained</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0:05:55</td>
<td>0:08:42</td>
<td>0:10:55</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0:30:01</td>
<td>0:44:06</td>
<td>0:55:21</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0:59:56</td>
<td>1:28:01</td>
<td>1:50:29</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>1:35:01</td>
<td>2:19:33</td>
<td>2:55:05</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>2:04:59</td>
<td>3:03:34</td>
<td>3:50:26</td>
</tr>
<tr>
<td>7</td>
<td>26.2</td>
<td>2:40:00</td>
<td>3:55:00</td>
<td>4:55:00</td>
</tr>
</tbody>
</table>
How to Make the Table

```matlab
1  % UAF/GI  Beyond the mouse, fall 2010, Ronni Grapenthin
2  % EXAMPLE: 2D matrix (Table), prints list of times that can be used for optimal
3  % Equinox 2011 preparation
4  % parameter: miles — miles you've run
5
6  function  pace_table = pacing_table(miles)
7
8  % Set up pacing table: Give miles as numbers and times as strings (requires a cell array,
9  % hence the curly braces)
10  pace_table = { 1   '0:05:55' '0:08:42' '0:10:55';
11    5   '0:30:01' '0:44:06' '0:55:21';
12    10  '0:59:56' '1:28:01' '1:50:29';
13    15  '1:35:01' '2:19:33' '2:55:05';
14    20  '2:04:59' '3:03:34' '3:50:26';
15    26.2 '2:40:00' '3:55:00' '4:55:00'};
16
17  % Since I'm lazy and didn't want to type all the miles, a mile does not equal the index,
18  % hence we'll have to do some math. Index is rounded number of miles divided by 5. Since
19  % Matlab indices start at 1, we have to add a 1. Otherwise everything smaller than 2.5 miles
20  % would result in an error
21  idx = round(miles/5)+1;
22
23  % lame output
24  pace_table(idx,:)
25  pause
26
27  % fancy output:
28  disp('-----');
29  disp('...miles.......record.......well.trained.......mildly.trained.');
30  disp('-----------------------------');
31  disp(pace_table(idx,:));
32  end
```

Listing 2.2: pacing_table.m
The Importance of Playing Around

• You will learn more if you spend time playing around with the computer, trying to make it do something interesting to you.

• You can start with the exercises, typing them from the lecture notes or even doing a copy and paste
  – You do have to watch out for apostrophes: the straight apostrophe and the curly ones (’’) are actually different characters!
  – Word processors today “help” you by automatically making curly apostrophes and quotation marks because it looks fancier.