Programming and algorithms

Marcello Vichi

Department of Oceanography
marcello.vichi@uct.ac.za

AOS

Outline

Programming languages

- A programming language is a set of structured instructions that tell the computer what to do. There have been many programming languages in the history of computer science. Some of them survived from the early 60's (FORTRAN) or evolved (ANSI C into C++ and C#) but only a few are very much used today in general applications: Python, C++, PHP, and Java.
- Why so many languages? (http://www.levenez.com/lang/)

High-level and low level languages

- FORTRAN, Python, C++, PHP, C#, Java, etc **are high-level languages**
- There are also low-level languages, sometimes referred to as machine languages or assembly languages. Loosely speaking, computers can only execute programs written in low-level languages. Thus, programs written in a high-level language have to be translated into something more suitable before they can run. (Useful resource: http://openbookproject.net/thinkcs/python/english3e/index.html)
- Almost all programs are written in high-level languages because of their advantages. It is much easier to program in a high-level language so programs take less time to write, they are shorter and easier to read, and they are more likely to be correct. Secondly, high-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications

The winning trend in computer programming

- To create and work with objects:
  - Reusable software components that model items in the real world
  - Meaningful software units: Date objects, time objects, invoice objects, audio objects, video objects, file objects, record objects, model grid objects, etc. These classes allow to define the rules controlling the "life" of these objects in the code
  - Any noun can be represented as an object
- Very reusable
- More understandable, better organized, and easier to maintain than procedural programming
- Favour modularity

Interpreters and compilers (I)

- The instructions for the computer need to be fed to the CPU. How does it happen? Through **interpreters** and/or **compilers**.[1]
- With an **interpreter**, the language comes as an environment, where you type in commands at a prompt and the environment executes them for you. For more complicated programs, you can type the commands into a file and get the interpreter to load the file and execute the commands in it. If anything goes wrong, many interpreters will drop you into a debugger to help you track down the problem.
- The advantage of this is that you can see the results of your commands immediately, and mistakes can be corrected readily.

[1] https://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/tools-programming.html

Interpreters and compilers (II)

- **Compilers** are rather different. You write your code in a file (or files) using an **editor** (that interpret ASCII characters, like textedit in windows). You then run the compiler and see if it accepts your program. If it did not compile, *grit your teeth* and go back to the editor; if it did compile and gave you an **executable program** (an *app*), you can run it either at a shell command prompt or in a debugger to see if it works properly.
- Not quite as direct as using an interpreter. However it allows you to do things that are very difficult or even impossible with an interpreter: writing code which interacts closely with the operating system; write very efficient code, as the compiler can optimize the code, which would not be acceptable in an interpreter. Finally, this is the way people create *apps* for various operating systems!
- As the edit-compile-run-debug cycle is rather tedious when using separate programs, many commercial compiler makers have produced **Integrated Development Environments** (IDEs for short).

The UNIX shell and the home directory

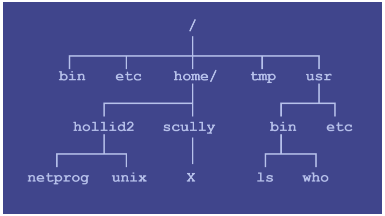
The language used in the terminal shell is an interpreted language.It allows you to interact with the file system and your data files. When the terminal is launched, it usually goes to your personal directory (the unix name for a folder), which is called your *home* (note that in WSL this is your windows home)

- All the users' directories are in /home

  ```
  /home/marcello
  /home/student
  ```

- It is the location where all your files go (organised into subdirectories). Location of many startup and customization files.
- It can be local (as in your case) or *mounted* from an external file server
- Run the command `cd` (change directory, by itself) to take you home from wherever you are

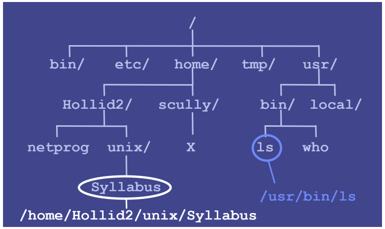
The UNIX filesystem



- Every file has a name and **file names** are case-sensitive! Unix file names can contain any characters (although some make it difficult to access the file) except the null character and the slash (/).
- A **directory** is a special kind of file - Unix uses a directory to hold information about other files
- Each file in the same directory must have a unique name. Files that are in different directories can have the same name. Files that start with a '.' are by default hidden
- The **filesystem** is a hierarchical system of organizing files and directories. The top level in the hierarchy is called the "root" and holds all files and directories. The name of the root directory is /

The UNIX pathname

- The **pathname** of a file includes: *the file name, AND the name of the directory that holds the file, AND the name of the directory that holds the directory that holds the file, AND the name of the . . . and so on up to the root*
- These pathnames are called **absolute pathnames**
- The pathname of every file in a given filesystem is unique
- You can use **relative pathnames**, that depend on the current working directory



```
$ cd /home/Hollid2
$ pwd
/home/Hollid2
$ ls unix/Syllabus
unix/Syllabus
$ ls X
ls: X: No such file or directory
$ ls /home/scully/X
/home/scully/X
```

Practicing is the only way forward

The UNIX commands are mnemonic, hence the only way to learn them is through practicing a few examples

- Read the quick introduction found here https://earth-env-data-science.github.io/lectures/environment/intro_to_unix.html
- Practice the tutorial on VULA (from Software Carpentry)

Example: a batch script to retrieve multiple files

- The example below is a program (*script*) written in *bash* (Bourn Again SHell, the default shell in linux) that uses the unix commands `seq` and `wget` to download monthly ocean colour data files from the ESA OC-CCI website for the years 2010-2015. This script is stored in an ASCII file that has been created with a text editor and named `multiple_oc-cci.sh`. Check the most recent version of the occci dataset.
-

```
#!/bin/sh
user="oc-cci-data"
passwd="ELaiWai8ae"
url="ftp://${user}:${passwd}@ftp.rsg.pml.ac.uk/occci-vXX/geographic/
    netcdf/monthly/chlor_a/"
years=$(seq 2010 2015)
echo ${years}
for y in ${years}
do
  wget -r ${url}${y}
done
```

The output

```
$ ./multiple_oc-cci.sh
2010 2011 2012 2013 2014 2015
--2017-02-08 23:03:26--  ftp://oc-cci-data:*password*@ftp.rsg.pml.ac.uk/occci-v3.0/geographic/netcdf/monthly/
     chlor_a/2010
           => 'ftp.rsg.pml.ac.uk/occci-v3.0/geographic/netcdf/monthly/chlor_a/.listing'
Resolving ftp.rsg.pml.ac.uk... 192.171.164.182
Connecting to ftp.rsg.pml.ac.uk|192.171.164.182|:21... connected.
Logging in as oc-cci-data ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD (1) /occci-v3.0/geographic/netcdf/monthly/chlor_a ... done.
==> PASV ... done.    ==> LIST ... done.
==> CWD (1) /occci-v3.0/geographic/netcdf/monthly/chlor_a/2010 ... done.
==> PASV ... done.    ==> LIST ... done.
--2017-02-08 23:03:29--  ftp://oc-cci-data:*password*@ftp.rsg.pml.ac.uk/occci-v3.0/geographic/netcdf/monthly/
     chlor_a/2010/ESACCI-OC-L3S-CHLOR_A-MERGED-1M_MONTHLY_4km_GEO_PML_OCx-201001-fv3.0.nc
           => 'ftp.rsg.pml.ac.uk/occci-v3.0/geographic/netcdf/monthly/chlor_a/2010/ESACCI-OC-L3S-CHLOR_A-MERGED-1
                M_MONTHLY_4km_GEO_PML_OCx-201001-fv3.0.nc'
==> CWD not required.
==> PASV ... done.    ==> RETR ESACCI-OC-L3S-CHLOR_A-MERGED-1M_MONTHLY_4km_GEO_PML_OCx-201001-fv3.0.nc ... done.
Length: 305695550 (292M)
aphic/netcdf/monthly/chlor_a/2010/ESAC   0%[
                                                                                   ]   1.13M   117KB/s    eta 57m
     50s
```

The use of wget -r creates a local directory structure equal to the one in the remote server

Matrix approach to scientific computing

- The previous script is a basic example of the *loop construct*, which is a sequential iterative operation on a given data type
- Matlab was one of the first languages to introduce complex data structures (such as vectors and matrices) as *atomic* data types in an interpreted language
- This allowed the user to work with them as if they were scalar variables and without the need for explicit looping (*a lot* more efficient)
- To appreciate the importance of this in the era of drag'n drop, graphical user interfaces and object-oriented programming, check out how low-level programming languages used to do the addition of a constant to a list of numbers (a vector)

The ~~good~~ old days - FORTRAN77 vs MATLAB

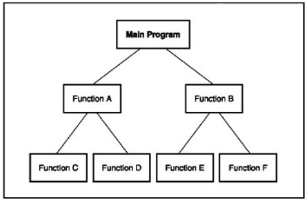
How to add 100 to the array (1,2,3,4,5,6,7,8,9,10)

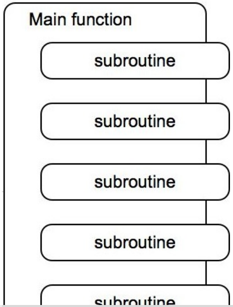
```
      PROGRAM ADDCNST
      PARAMETER (N=10)
      PARAMETER (C=100.0)
      REAL A(N)
      DO 33 I=1,N
      A(I) = I
33    CONTINUE
      DO 66 I=1,N
      A(I) = A(I)+C
66    CONTINUE
      WRITE(6,*) A
      END
```

```
A = 1:10;
A = A+100
```

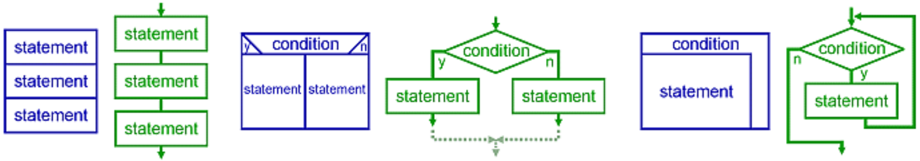
Structures of a program

- Big codes are divided in smaller portions to improve usability and readability. They have a **main program** and **(sub)procedures**
- Procedures may be
    - **subroutines** smaller portions of the code that do certain things and can be called with specific arguments. They are used to structure the code in meaningful self-contained parts. Both the input and output arguments can be complex objects with multiple information
    - **functions** these are portions of the code that return *at least* one result. They are meant to be used many times, as for instance the intrinsic mathematical functions (`sin`, `cos`, `sqrt`, etc.)





Structured programming (Wikipedia)



Sequence	ordered statements or subroutines executed in sequence
Selection	one or a number of statements is executed depending on the state of the program `if..then..else..endif`.
Iteration	a statement or block is executed until the program reaches a certain state, or operations have been applied to every element of a collection. This is usually expressed with keywords such as `while`, `repeat`, `for` or `do..until`.
Recursion	a statement is executed by repeatedly calling itself until termination conditions are met.

Algorithms

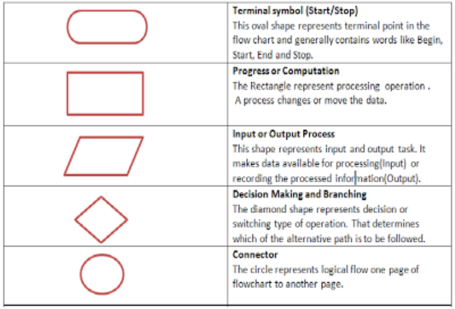
Definition

An algorithm is a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminate at some point *(http://mathworld.wolfram.com/Algorithm.html)*

- The word "algorithm" is a distortion of al-Khwārizmī, a Persian mathematician who wrote an influential treatise about algebraic methods
- The process of applying an algorithm to an input to obtain an output is called a computation
- Algorithms are presented as flow charts and usually expressed in meta-language terms
- This is done because they have to be generic and independent of the specific programming language

Flow charts

- A flowchart is a graphical representation of an algorithm
- It uses frame symbols to indicate certain basic operations (simple algebraic, input/output, decision making, etc.)
- Symbols are connected to each other with arrows showing the flow and the meta-language operations are indicated inside the frame

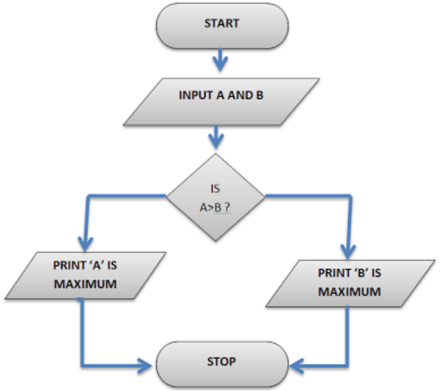


⬭	**Terminal symbol (Start/Stop)** This oval shape represents terminal point in the flow chart and generally contains words like Begin, Start, End and Stop.
▭	**Progress or Computation** The Rectangle represent processing operation . A process changes or move the data.
▱	**Input or Output Process** This shape represents input and output task. It makes data available for processing(Input) or recording the processed information(Output).
◇	**Decision Making and Branching** The diamond shape represents decision or switching type of operation. That determines which of the alternative path is to be followed.
○	**Connector** The circle represents logical flow one page of flowchart to another page.

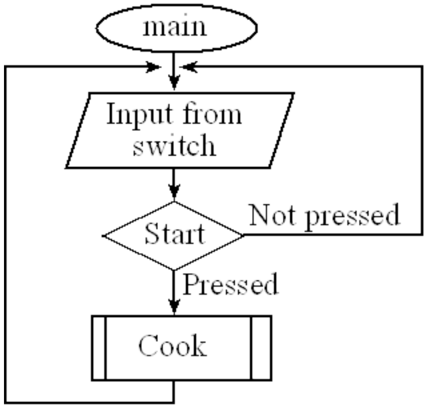
A trivial example

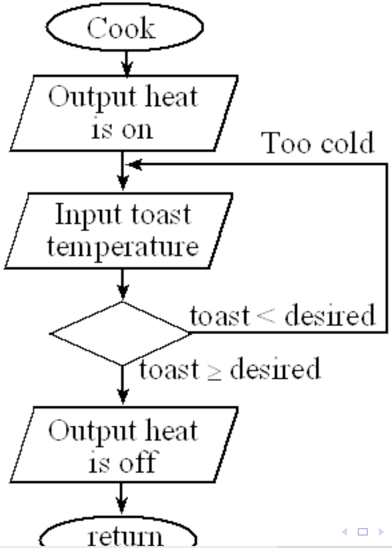
```
Meta-language Algorithm

Step 1: Start
Step 2: Input the values of
        A and B
Step 3: Check if a>b,
        - if yes then print
          "A is maximum"
        - if no then print
          "B is maximum"
Step 4: Stop
```



Another one with structured programming: Cooking a toast...


main
Input from switch
Start
Not pressed
Pressed
Cook


Cook
Output heat is on
Too cold
Input toast temperature
toast < desired
toast ≥ desired
Output heat is off
return