12.010 Computational Methods of Scientific Programming

Lecture 14: Compiled languages

Summary

- Start compiled languages:
- Getting compilers (Fortran/C)
- How they work (compile/link/run)

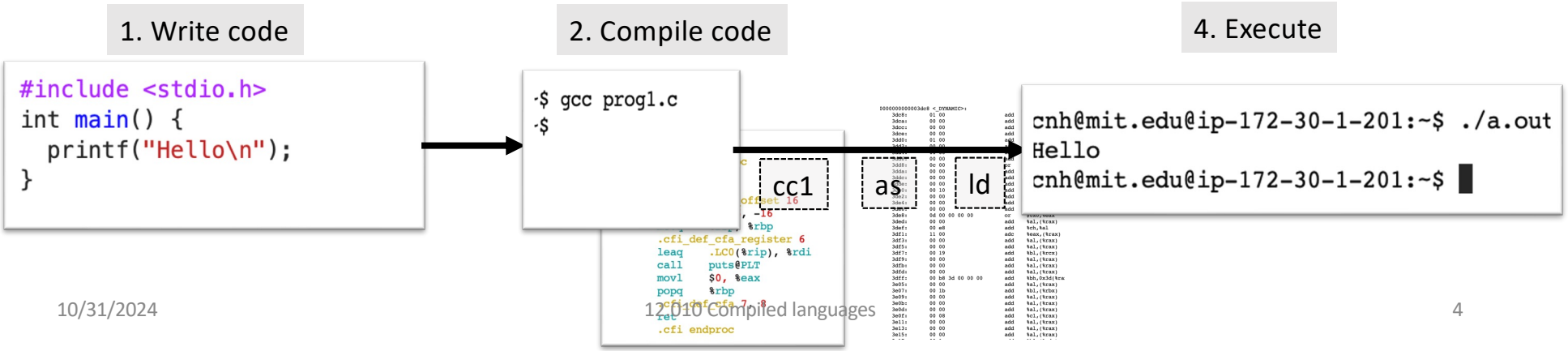
- Goal is to give some basic familiarity and see how key foundational concepts (loops, conditionals, functions, data structures, I/O) look and behave.
  - Fully learning each language in detail takes time and practice.

Compiled languages:

- Is it worth learning more than Python?
  - Great question!
  - Python is getting remarkably capable e.g.
    - https://veros.readthedocs.io/en/latest/
      - Global ocean model (https://dionhaefner.github.io/2021/04/higher-level-geophysical-modelling/) all in Python, uses various packages (NUMBA, JAX) to allow
        - NUMBA provides JIT (just in time) compilation for Python, which can make it perform well.
        - JAX provides a plugin for GPUs
    - In principle could just work in Python
  - Lots of existing tools in Fortran, C, C++
    - Exoplanets, seismic, fluids, ML, materials, econ etc…
    - Knowing some can help make sense of these tools
  - **Also** - Learning a little about other languages can help thinking overall, enhance understanding of concepts versus “implementation”.

Compiled languages:

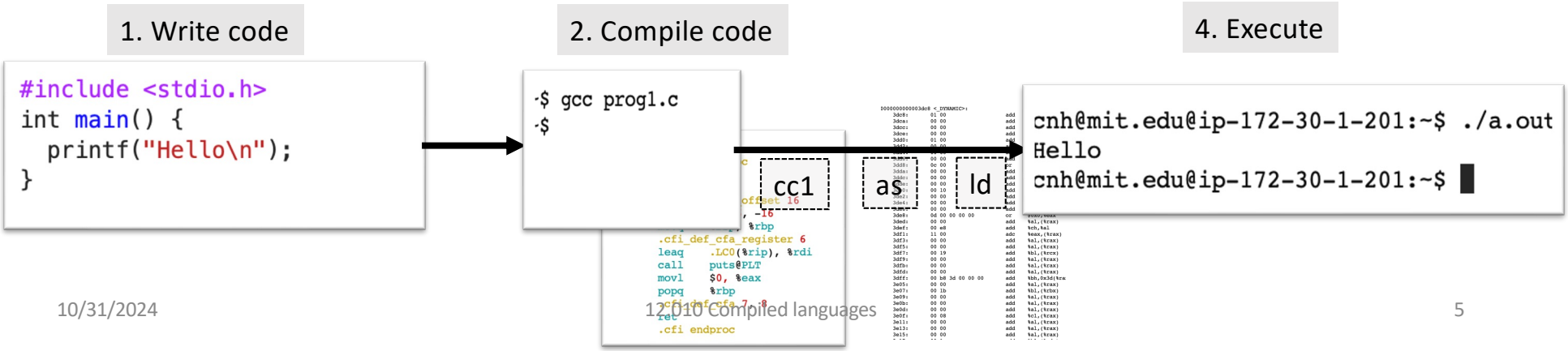
- What does “compiled language” mean
  - Using a compiled language means
  1. Code (C, Fortran, Java, C++) is written to a file (or files) in a human readable form
  2. The file(s) of code are processed by a special program, called a compiler.
  3. The compiler produces a separate file of output, called an executable.
  4. The executable is a program that can be run to perform the steps written in the original program



Compiled languages:

1. Writing code
   - uses a programming oriented editor e.g. vs-code, vi, emacs, nano, nedit etc….

2. Compile and link code
   - “compiles” one or more files into “object” code
     - foo.c ➔ foo.o, bar.c ➔ bar.o
   - “links” .o files to create executable (default name a.out ?)
     - *.o ➔ a.out



Compiled languages v. interpreted - I

- What are some differences between compiled and notebook/Python (i.e., interpreted languages).
  - Compiled code can be shared as a single binary file (for particular CPU + OS) and will execute.
  - Interpreted code requires an “interpreter” program (e.g., Python)
  - Developing with compiled code is generally more “pedantic”. Compiled languages quite often have more syntax rules, and the workflow is less interactive (harder to see what is going on, debugging requires more planning).
  - Compiled code can often produce faster, more efficient and smaller programs (not guaranteed!)

Compiled languages and uses

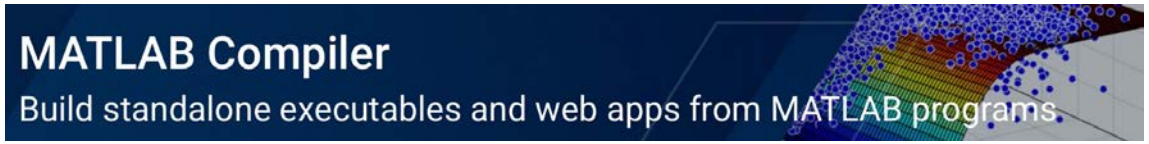
- Top 20 languages (https://www.tiobe.com/tiobe-index/) for 2021 has eleven compiled and nine interpreted
  - C(2), Java(4), C++(3), C#(5), VB(7), AS(10), Go(11), Swift(15), Fortran(16), Pascal(?)
  - Python(1), Javascript(6), SQL(9), PHP(8), MATLAB(14), R(17), Ruby(19), Perl(?)
- Compiled used for e.g.
  - Operating system, Weather forecast models, planetary, stellar, geophysics, astro, mech, particle, real-time control etc…
- Interpreted also used, and growing
- Line between compiled v. interpreted can be blurred
  - Can get interpreters for compiled langs
  - Can “compile” interpreted langs!

Oct 2023	Oct 2022	Change	Programming Language	Ratings	Change
1	1		Python	14.82%	-2.25%
2	2		C	12.08%	-3.13%
3	4	↑	C++	10.67%	+0.74%
4	3	↓	Java	8.92%	-3.92%
5	5		C#	7.71%	+3.29%
6	7	↑	JavaScript	2.91%	+0.17%
7	6	↓	Visual Basic	2.13%	-1.82%
8	9	↑	PHP	1.90%	-0.14%
9	10	↑	SQL	1.78%	+0.00%
10	8	↓	Assembly language	1.64%	-0.75%
11	11		Go	1.37%	+0.10%
12	23	⇈	Scratch	1.37%	+0.69%
13	18	⇈	Delphi/Object Pascal	1.30%	+0.46%
14	14		MATLAB	1.27%	+0.09%
15	15		Swift	1.07%	+0.02%
16		↑	Fortran	1.02%	+0.23%

Screenshot

Compiled languages v. interpreted - II

- Quite a few languages are a bit of both – compiled v interpreted workflow is different, but language can be the same
  - For example, Matlab has a compiler that can be used to generate executables that can be run on systems without Matlab installed.



  - Fortran and C can be written in a Notebook

```
[4]: integer I
     I=7

[5]: WRITE(*,*) I
     7

[ ]:
```

```
[5]: #include <stdio.h>
     int main(){
         printf("hello\n");
     }
     hello
```

https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

We will focus on compiled languages that are commonly used in science

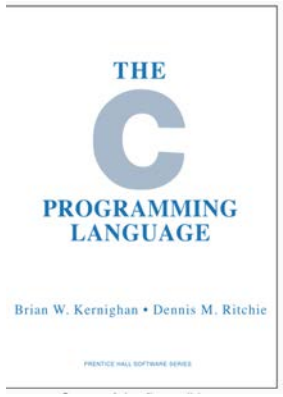
- Fortran, C, C++
- Some common compiled language aspects
  - Much more use of explicit “types”
  - More fiddly syntax
  - Edit, compile, run workflow is similar
    - make and build tools etc….
- Some differences between Fortran, C, C++
  - Syntax is different, although concepts (loops, functions, conditionals) the same
  - Fortran has a numerical/scientific feel builtin (for example native multi-dimensional arrays)
  - C, C++ are a little more general computing oriented (pointers, memory management etc…)

Getting started with C

- Quick look at basics of language
- compiler, make tools etc…
- Try compiling, running, modifying some examples

Getting started with C

- C programming language originally designed in late 1960s, early 1970s.
  - first popularized as “Kernighan and Ritchie” (K&R) C in 1978. Part of UNIX OS,
  - much of UNIX (and subsequently Linux, Windows, MacOS etc..) in C
  - 1989 introduced ANSI C, a more official standard (with standards board etc..).
  - Latest is C17 (around 2017)
  - C is very portable, and very widely used. Almost every flavor of hardware has a C compiler.
    - Not all hardware supports all features
    - Features beyond basic library are in a standard library (https://en.wikipedia.org/wiki/C_standard_library).



Getting started with C

A simple C program.

```c
#include <stdio.h>
#include <math.h>
int main()
{

 /* Write Hello */
 printf("Hello\n");

 /* Write Hello and the value of PI */
 /* Two statements separated by ;   */
 /* PI is defined in math.h         */
 /* without math.h you get an error */
 /* at compile time if you try to   */
 /* print PI.                       */
 fprintf(stdout,"Hello\n");   fprintf(stdout,"pi == %f\n",M_PI);

 return 1;
}
```

"Included" header files - cpp

All programs start from the main() function.

Executable statements always end in ";"

Comments /* .. */ or //

printf(…), fprintf(…) are function calls (for I/O) from standard library.

stdout, M_PI are from "header"

All functions (including main()) can return some result

Getting started with C

Everything in C needs to "typed"

Key primitive types
int, float, double, char, * (pointer type).

```c
#include <stdio.h>
int main() {
  short a;
  long b;
  long long c;
  long double d;
  double e;
  float f;
  int g;
  char h;
  void *p;

  printf("size of short = %lu bytes\n", sizeof(a));
  printf("size of long = %lu bytes\n", sizeof(b));
  printf("size of long long = %lu bytes\n", sizeof(c));
  printf("size of long double = %lu bytes\n", sizeof(d));
  printf("size of double = %lu bytes\n", sizeof(e));
  printf("size of float = %lu bytes\n", sizeof(f));
  printf("size of int = %lu bytes\n", sizeof(g));
  printf("size of char = %lu bytes\n", sizeof(h));
  printf("size of pointer = %lu bytes\n", sizeof(p));
  return 0;
}
```

Getting started with C

- Generally compiled languages are “fussier” over syntax.

```
cnh@mit.edu@ip-172-30-1-201:~$ gcc foo.c
foo.c: In function ‘main’:
foo.c:4:2: error: ‘i’ undeclared (first use in this function)
    4 |  i = 1
      |  ^
foo.c:4:2: note: each undeclared identifier is reported only once for each function
it appears in
foo.c:4:7: error: expected ‘;’ before ‘printf’
    4 |  i = 1
      |      ^
```

Compiler gives error, because I didn’t declare the variable “i” to be an integer.

```
cnh@mit.edu@ip-172-30-1-201:~$ gcc foo.c
foo.c: In function ‘main’:
foo.c:6:2: error: expected ‘,’ or ‘;’ before ‘printf’
    6 |  printf("Hello\n");
      |  ^~~~~~
```

Compiler gives error, because I forgot a “;” at the end of the line.

- The more exacting syntax rules can make writing code more awkward, but they help the compiler generate optimized code.

Getting started with C

- Conditionals in C

```c
#include <stdio.h>
#include <errno.h>

int main(){

  int i = 0;

  /* == returns logical result, proper conditional */
  if ( i == 0 ) {
    printf("test 1: i is set to %d\n",i);
  }

  /* Using = not == for equality tests is a common typing mistake in C    */
  /* Using = is valid syntax but it doesn't mean what you think it means! */
  /* i = 0 is false ( it returns 0 ) by definition, irrespective of the   */
  /* value of i.                                                          */
  if ( i = 0 ) {
    printf("test 2: i is set to %d\n",i);
  }

  /* i = 1 is true ( it returns non-zero ) by definition, irrespective of */
  /* the value of i.                                                      */
  if ( i = 1 ) {
    printf("test 3: i is set to %d\n",i);
  }

}
```

== is used to test for equality.

= is used for assignment. It returns the value assigned – so can be false (==0) or true (==1) depending on assigned value.

If statement syntax is

```
if ( condition ) {
   expression;
} else {
   expression;
}
```

else part is optional.

Getting started with C

- Loops in C

```c
#include <stdio.h>
int main() {
    int i;
    int L=10;
    // for loop condition evaluates at start of loop iteration
    // for loop has ( initial; condition; loop "increment" ) { }
    // syntax.
    for (i=0; i<L; i=i+1){
        printf("for i=%d\n",i);
    }
    // while loop condition evaluates at start of loop iteration
    while (i>0){
        printf("while i=%d\n",i);
        --i;
    }
    // do while loop condition evaluates at end of loop iteration
    do {
        printf("do while i=%d\n",i);
        ++i;
    } while (i<L);
    // do while loops always execute once
    do {
        printf("do while, 2 i=%d\n",i);
        ++i;
    } while (i<L);

    return 0;
}
```

Python – loops indented.

C syntax

```
for ( start; test; iteration_update ) {
    expression;
}
```

Getting started with C

- Arrays in C

```c
#include <stdio.h>
// CPP macro
#define L 10
int arrfunc(){
    int AI[L];
    int i;
    i=0;
    for (i=0;i<L;++i){
        AI[i]=i;
    }
    for (i=0;i<L;++i){
        printf("AI[%d]=%d\n",i,AI[i]);
    }
    return 0;
}
```

Arrays in C – simple way with [] notation.

Array sizes in simple way need to be set at “compile” time.

Arrays can be multi-dimensional.

Getting started with C

- Arrays in C are pointers

```c
#include <stdio.h>
// CPP macro
#define L 10
int arrfunc(){
    int AI[L];
    int *p=AI;
    int i;
    i=0;
    // 0 based indexing
    for (i=0;i<L;++i){
        *(p+i)=i;
    }
    for (i=0;i<L;++i){
        printf("AI[%d]=%d\n",i,AI[i]);
    }
    return 0;
};
```

Arrays in C are one example of "pointers"

Pointers are often used in C and can be a bit mysterious to start.

In essence they are a "pointer" to some block of memory.

Getting started with C

- Hands on
  Lec13-c-basic-hello.ipynb
  But need C Kernel installed for this to run (and maybe only possible on Unix and Mac)
- When this code is loaded (at least in vscode) there should be a prompt to install the C/C++ extension pack.
- For Macs:

xcode-select --install

(invokes getting the command line)

https://github.com/fxcoudert/gfortran-for-macOS/releases

(Then use gcc).

- Run in athena.dialup.mit.edu (needs two-factor authentication).

MIT OpenCourseWare

https://ocw.mit.edu

12.010 Computational Methods of Scientific Programming, Fall 2024