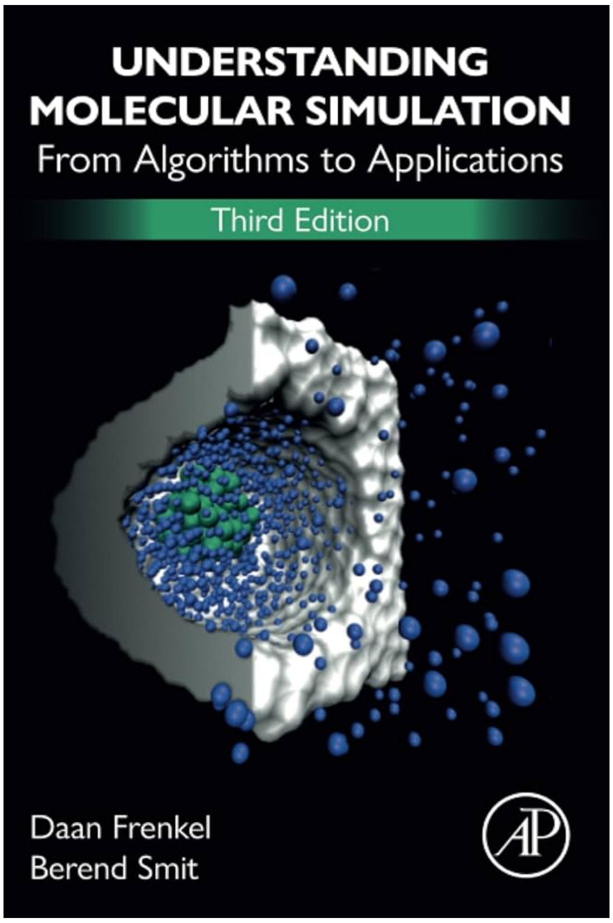
CH-420

Understanding Advanced Molecular Simulation



Block 0

Computational Carpentry - C tutorial

1 C tutorial

1.1 Summary of what you will need to do across this course

1. Make modifications on the source code (program.c) in each exercise when required, to add the necessary equations or variables needed to run the code.
2. Run `make clean` on the source directory ('Source').
3. Run `make` on the source directory ('Source').
4. Run the program. This command will ask you for input data sometimes, and generate datasets that you will use to plot. You can run the program by accessing the folder 'Run' and typing either one of the options:
    - ../Source/program
    - ./run
5. Plot the data with your plotting tool of choice (you can write your own script but we provide examples).

Read further for more details on each.

1.2 C hierarchy and structure

Running a C programs requires writing the code and compiling the code, i.e., translating human-readable source code written in a high-level programming language (such as C, C++, Python, etc.) into machine-readable binary code that can be executed by a computer's processor.
First, let's break down the hierarchy in a C project and the roles of Makefile, source code files (.c files), and executable files (run files).

1.2.1 Source Code Files (.c files)

- Source code files contain the actual code written in the C programming language.
- Each .c file typically corresponds to a specific module or component of the software.
- These files contain function definitions, variable declarations, and other necessary code to implement specific functionalities.

1.2.2 Header Files (.h files)

- Header files contain declarations of functions, variables, and types that are shared across multiple source code files.

- They usually include function prototypes, type definitions, and preprocessor directives like #define statements (for constants and macros).
- Header files are #included in source code files to make the declarations available during compilation.
- The equivalent of a header file (C) in python is usually a module or a package that can be imported directly in the python code where one wants to use the functionalities.

1.2.3 Makefile

- A Makefile is a script that automates the build process of a project.
- It specifies how source code files should be compiled and linked to create executable files.
- Makefiles define rules and dependencies for building the project, such as which source files to compile, compiler flags, and how to link object files.
- Makefiles typically consist of targets, dependencies, and commands to execute.
- Make is a build automation tool that reads the Makefile and executes the specified commands to build the project.

1.2.4 Executable Files (Run Files)

- Executable files are the final output of the compilation process.
- They are generated by compiling and linking the source code files.
- Executable files contain machine code that can be directly executed by the computer's processor.
- In C projects, the executable file is often generated with a name specified in the Makefile or by default (e.g., a.out on Unix-like systems).

1.2.5 Examples of hierarchy

The hierarchy we use in the course is the following:
Inside each exercise you will find folders "Source" and "Run". Inside "Source":

- Makefile: Controls the compilation process.
- source code files (.c files).

Inside "Run": - run: Contains the executable files generated after compilation. Alternatively in some exercises you could manually run the program (from compiled program.c) with `../Source/program` - plot: Different plot files will be available (in python and gnuplot). Python is preferable. You can also write your own script.

1.3 How to compile a C code?

To compile a C code, you typically use a C compiler such as GCC (GNU Compiler Collection) or Clang. Here's a general overview of the process:

1. Write Your C Code: First, write your C code using a text editor or an integrated development environment (IDE). Save the file with a .c extension.
2. Open Terminal or Command Prompt: Open a terminal or command prompt window.
3. Navigate to the Directory Containing Your C File: Use the cd command to navigate to the directory where your C file is located.
4. Compile Your C Code: Use the C compiler to compile your C code into an executable file. The basic syntax for compiling a C file with GCC is: `gcc source_file.c -o output_file`
   - It might be useful to include -lm in the end to link your program with the math.h library so that the calls to math functions are resolved. `gcc source_file.c -o output_file -lm`
5. Running the executable: `./output_file`

If there are any errors during compilation, the compiler will display error messages that you'll need to address before trying to compile again. Additionally, you can use compiler flags to specify options such as optimization levels, debugging information, or warning levels.
To facilitate the process of compiling a code, and automate the 4th step, it can be helpful to use `make`. Using make simplifies the compilation process by automating the steps needed to build a project. Here's how you can compile a C code using make:
The comments in italic are what you actually need to do in this course

- Write Your C Code: First, write your C code and save it with a .c extension. For example, let's say your C file is named my_program.c. This step is already done for you in this course but *you need to make modifications according to each exercise!*
- Write a Makefile: Create a Makefile in the same directory as your C file. The Makefile specifies the rules and dependencies for building your project. This step is already done for you!
- *run `make clean` on your terminal to avoid errors*
- *run `make`* to compile your code

That's it! You have compiled your C code using make. If there are any errors during compilation, make will display error messages that you'll need to address before trying to compile again. Additionally, you can customize the Makefile to include additional rules or dependencies as needed for your project.

1.4 Inside the source code

You will need to understand and modify the source codes. Here we list some information for code comprehension that might help you.

1.4.1 Comments

- Comments in C are done via the double inverse slash \\.

1.4.2 Include header files

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "myheader.h"
```

- These lines include necessary header files for input/output operations (stdio.h), dynamic memory allocation (stdlib.h), and mathematical functions (math.h). It serves the same purpose as importing a library in python, but with different mechanisms.
- The last line `#include "myheader.h"`refers to a custom header file in the current C source file.

1.4.3 Defining constants or macros

- define can improve code readability by giving meaningful names to magic numbers or repetitive code snippets. It also allows for easy modification of values or code snippets because you only need to change them at one place in the code.

```
#define CONSTANT 10000
#define MAX_NUMBER_STEPS 20000
```

- This line defines a constant CONSTANT with a value of 10000.

  ```
  #define SQUARE(x) ((x) * (x))
  ```

- This line defines a macro named SQUARE that squares its argument x. Whenever SQUARE(somevalue) appears in the code, it gets replaced by (somevalue * somevalue) during preprocessing.

1.4.4 Declaring variables

In C and many other programming languages, int, double, char, etc., are data types used to define variables. Each data type represents a different kind of value that a variable can hold. Here's a brief overview of when you might use each of these data types:

```
int age = 30;
```

- int: Stands for integer. It is used to store whole numbers (positive or negative) without any fractional part. For example, int might be used to represent quantities, counts, or indices in a program.

  ```
  float temperature = 98.6;
  ```

- float: Stands for floating-point. It is used to store real numbers with a fractional part. float provides less precision compared to double, but it requires less memory. It's commonly used when memory efficiency is critical, such as in embedded systems or when working with large arrays of floating-point numbers.

  ```
  double pi = 3.14159;
  ```

  ```
  double Example[parameters];
  ```

- double: Stands for double precision floating-point. It is used to store real numbers (positive or negative) with a fractional part. double provides more precision compared to float. It's commonly used for representing measurements, coordinates, or any other real-world values that require precision.
- in the last example, we are declaring an array called Example that stores the information for each parameter. It is like an array in python with size n, where n is the number of the parameters.
- you can initialize temporary variables with, e.g., `double tmp;`

  ```
  char grade = 'A';
  ```

- char: Stands for character. It is used to store single characters, such as letters, digits, or special symbols. In C, characters are enclosed in single quotes ('). char variables are commonly used in string manipulation and character-based operations.

  ```
  short int smallNumber = 1000;
  long int largeNumber = 1000000;
  ```

- short and long: These are modifiers used with int to specify the range of values a variable can hold. short int typically uses less memory than int but can represent a smaller range of values, while long int can represent a larger range of values but may use more memory.

```
FILE *FilePtr;
```

- FilePtr is a file pointer used for file operations, i.e., parse, open, close, save etc. files with the desired data.

These are some of the fundamental data types in C, and each is used in different situations based on the requirements of the program and the range/precision of values needed to be stored.

1.4.5 Functions in C

```
int main(void)
{
    ...
}
```

- This code block declares the main function which serves as the entry point of the program. It doesn't take any arguments (void).

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    char hello[13] = "Hello, World!";
    printf("%s", hello);
    return 0;
}
```

- This code block prints "Hello, World!" using a string (with 13 characters, i.e., when char is used in conjunction with square brackets `[13]`, it indicates that you're declaring an array of characters, not just a single character).
- `return 0;` statement indicates that the program execution was successful, and it returns 0 to the operating system.
- `int main()` historically indicates that main takes an unspecified number of arguments. It's a legacy feature inherited from older versions of C, and omitting parameters in a function declaration implies that the function can accept any number of arguments. However, this form is discouraged in modern C programming because it doesn't explicitly state that main takes no arguments Another example:

```
float equation(float v1, float v2, float v3) {
    float V1;
    V1 = v1*v2/v3;
    return V1;
}
```

1.4.6 Reading input variables

```
printf("Number of steps (2-10000) ? ");
fscanf(stdin,"%d",&NumberOfSteps);

printf("Pressure(bar)? ");
fscanf(stdin,"%lf",&Pressure);
```

- These lines prompt the user to input the number of steps and the pressure (in bar). The values are read from standard input (stdin) using fscanf.

1.4.7 Conditional statements

```
if (NumberOfSteps < 2 || NumberOfSteps > MAX_NUMBER_STEPS || Pressure < 0.5||
Pressure >= 2)
{
    printf("Input parameter error, should be\n");
    printf("\tNumberOfSteps (2-%d)\n", MAX_NUMBER_STEPS);
    printf("\tPressure (0.5 - 2 bar)\n");
    exit(0);
}
```

- This conditional statement checks if the input parameters (NumberOfSteps and Pressure) are within the specified ranges. If they are not, an error message is printed, and the program exits using exit(0) (this is like `sys.exit()` in python).
- || is used to add conditions to the conditional statement, with its logical syntax meaning "or".

1.4.8 Equations

```
#include <stdio.h>
#include <math.h>

int main() {
    double x = 2.5;
    double y = 3.0;
    double z = 1.2;

    // Various mathematical operations
    double result = sqrt(pow(x, 2) + exp(y) - log(z) * sin(x + y) / cos(z));

    printf("Result: %f\n", result);

    return 0;
}
```

- `sqrt`: Square root function.
- `pow`: Exponentiation function.
- `exp`: Exponential function.
- `log`: Natural logarithm function.
- `sin`: Sine function.
- `cos`: Cosine function.
- when using `double result = ...` you are declaring the variable named result and directly computing its value via an equation.
- This example calculates the value of a complex mathematical expression involving square root, exponentiation, exponential function, natural logarithm, sine, and cosine.

1.4.9 Loops

```
double Example[NumberOfSteps];
Normalize = 0.0;
for (i = 0; i < NumberOfSteps; i++)
{
    tmp = sqrt(pow(x, 2) + exp(y) - log(z) * sin(x + y) / cos(z));
    Example[i] = tmp;
    Normalize += tmp;
}
```

- This loop calculates, for each step in NumberOfSteps, the resulting value for the example equation and accumulates it in Normalize (this is a sum of all the results). It also stores the result for each step in an array called Example of size=NumberOfSteps, i.e., it iterates from 0 to NumberOfSteps - 1, calculating sqrt(pow(x, 2) + exp(y) - log(z) * sin(x + y) / cos(z)) and storing it in Examples[i].

```c
#include <stdio.h>

int main() {
    int i = 1; // Initialize loop control variable

    while (i <= 5) { // Loop condition: execute while i is less than or equal to 5
        printf("%d\n", i); // Print the current value of i
        i++; // Increment i for the next iteration
    }

    return 0;
}
```

- In this example, the loop control variable i is initialized to 1. The while loop executes as long as the condition i <= 5 is true. Inside the loop, the current value of i is printed, and then i is incremented by 1 (i++) to prepare for the next iteration. When i becomes greater than 5, the condition i <= 5 becomes false, and the loop terminates.

1.4.10 Handling files

```c
FilePtr = fopen("results.dat", "w");
for (i = 0; i < NumberOfSteps; i++)
    fprintf(FilePtr, "%d %f\n", i, Example[i] / Normalize);
fclose(FilePtr);
```

- These lines open a file named "results.dat" for writing. Then, it writes the index of each step (i) along with its corresponding normalized value (Example[i] / Normalize, i.e., the value divided by the sum of all values) into the file. Finally, it closes the file.