Reading Material

From *Computational Physics*:
- Chapter 1, §1.6

From *Teach Yourself C*:
- Chapter 1, §1.1 - 1.6
- Chapter 1, All examples and Exercises
A. Code Components

Every C code consists of:
1. a list of one or more \texttt{#include} statements for header files,
2. one or more functions that include statements that manipulate variables,
3. program comments.

(1) \textit{Header} files include all the information needed by the compiler to handle the functions and their statements correctly. There is a number of standard header files, which we will encounter along the way. One example is stdio.h (handling standard Input/Output) and the way such files are included in the code is:

\begin{verbatim}
#include <stdio.h>
\end{verbatim}

(2) \textit{Functions} are code structures similar to mathematical functions, in the sense that they take values of parameters as input and return the value of the function as an outcome.

Both the input parameters and the function outcomes are variables, which can be of different types. The first line of a function consists of the declaration of the type of its outcome, the function name, followed by a set of parentheses containing the declarations and names of the input variables, followed by a set of curly brackets containing the list of statements:

\begin{verbatim}
Type-of-function-output Name-of-function (Type-of-input-variable Name-of-input-variable, ... ) {
  ...  
  return output-variable or value  
}
\end{verbatim}

All C codes consist of at least one function called \texttt{main}: this is where execution of the code starts. The usual conventions for the type and input variables of \texttt{main} are: integer (\texttt{int}) and none, respectively. This is how the \texttt{main function} is declared:

\begin{verbatim}
int main() {  
    ....  
    return 0;  
}
\end{verbatim}

NOTE: \texttt{int} and the \texttt{return} line for the \texttt{main} function can be omitted.
(3) *Comments* are a necessary component of computer codes! They can be included *anywhere* in the code as text enclosed in: /* comment */

**B. Variables**

*Variables* are used to identify named pieces of memory that hold a value; that value is the value of the *variable*.

*Variables* can have different types.

They usually identify numbers and their type can be:

- integer: `int`
- real of single precision: `float`
- real of double precision: `double`

They can also identify single characters and their type is `char`.

*Variables* MUST BE DECLARED in the very beginning of the function where they are used. Values are assigned to *variables* with the use of “=”:

```c
int main () {
    float in, cm;
    int length;
    length= 3;
    in = 2.54;
    return 0;
}
```

*Variables* can also be initialized at their declaration line:

```c
int main () {
    float in=2.54, cm;
    int length=3;
    return 0;
}
```

Note that capital letters matter in C, so “A” and “a” are two different variables. However, relying on this is very dangerous because bugs can be introduced through *typos*!
Data types can be altered within codes. For example, if variable `i` has been declared as an integer, then it can be changed to data type `double` with the statement: `(double) i;`

C. Arithmetic Expressions

Values can be assigned to a variable through arithmetic calculations: additions, multiplications, power, etc. For all of these operations, there are standard symbols that must be used:

- addition: `a+b`
- subtraction: `a-b`
- multiplication: `a*b`
- division: `a/b`
- modulus: `a%b` (gives the remainder of the division)
  (NOTE: this is used only with integer-type variables)

It is very important to pay a lot of attention to the order of the various arithmetic operations:

```
a+b*c is different than  (a+b)*c
4+2*5 = 14 and (4+2)*5 = 30
```

* / % are higher in precedence than + and −. It is strongly encouraged that parentheses are used to make sure a sequence of operations calculates what you intended!

NOTE: Symbols of the same precedence are applied starting from the left-most operation.

Arithmetic expressions can be used to change the values of `variables`: e.g., `a=a+1`. Remember not to confuse value assignment lines with equations!
D. Screen Input/Output

Output can appear on the computer screen with the use of `printf()` and Input can be inserted from the screen (the keyboard really!) with the use of `scanf()`.

Before using any of the two remember to include the I/O header file, starting every program with `#include <stdio.h>` (it appears before the `main` function).

- Text output is included in quotes:
  ```c
  printf(“Welcome to PHYS-252”)
  ```

- Including “\n” inserts a line-feed:
  ```c
  printf(“Welcome to PHYS-252 \n Hope you’ll enjoy it!)
  ```
  NOTE: (1) Spaces are important (2) always include \n at the end of any output.

- To print the value of a variable (number or characters) a **format specifier** is required along with the variable or a value directly:
  ```c
  printf(“There are %i pages in this textbook; its cost is %f and the first letter of its author is %c. \n”,843,29.95,author-initial)
  ```

- To assign a value from the keyboard to a variable, i.e., read an input value, a format identifier is used as when printing, followed by the name of the variable preceded by &. The reason for this will be explained later. An example:
  ```c
  scanf( “%f”,&cost)
  ```

- To read the value of a variable of data type double, the identifier must be `%lf`

- The most important format identifiers are:
  ```c
  %i or %d for integers
  %f for float
  %lf for double
  ```
Programming Hints

- Think and create a basic “outline” for your code before you start programming.
- Include as many comments as possible explaining what each and every part of your code does.
- Break medium and large programs into smaller ones using many functions and separate pieces of code.
- Use variable names that make sense in English - even though this tends to make them long - they help you remember what the use and meaning of the variable are.
- Declare all the variables you are using.
- Avoid mixing different types of variables in arithmetic expressions. Instead convert variables to the same type, whenever possible.
- Remember to use parentheses to make clear what is the order of arithmetic operations.
- Never trust your code. Assume there are mistakes and try to think of ways to test it with simple examples and calculations, for which you can figure out the result independently.
- Remember to always include “;” at the end of every code line except for “# include” lines, and the opening/closing lines of functions.
Visual C++ Tutorial

Visual C++ provides us with a windows-based environment for compiling, linking, running, debugging programs written in C or C++. A single source-code file (or set of files) belong to a workspace (.dsw files) that needs to be set up before anything else. Once a workspace is opened, (i) code files can be created within the Visual C++ environment, in which case they are automatically added to the Project/Workspace, or (ii) code files that are already saved in the directory of the Workspace can be added to the current Project.

1. Double-click on the “Visual C++” icon on your desktop.

2. To access an existing Workspace:
   • Click on File, then on Open Workspace
   • Use Browse to find the directory with the workspace, then double-click on the .dsw file you are looking for.

3. To open a New Workspace/Project:
   • Click on File, then New
   • Click on Projects Tab, Win32 Console Application, then choose the directory for the new workspace, and type the name of the project (will also be the name of the workspace).
   • Click on Finish, and then OK.
   • Each exercise should be developed as a separate workspace and in a separate sub-directory. All sub-directories should be in a directory named after your last name.

4. To open a new source code file:
   • Click on File, then New
   • Click on Source C/C++ file, then type the name of the file (e.g., name-of-the-code.c).
   • An editor opens where you can write your program.

5. To add files to an existing Project/Workspace:
   • Copy files to the directory of the workspace
   • Click on Project, then on Add to Project
   • Choose the files to be added
6. Compiling and Running a code (see the series of icons on top right):

- *stack of papers*: compiles the C code you are editing
- *double arrow image*: links the whole Project
- *red exclamation mark*: runs the code from the beginning
- *sheet with an arrow*: runs from the position of a stop mark
- *hand*: puts a stop mark at the line of code you are editing
- A small screen at the bottom shows the outcome of the compilation and linking process.