Basics of C

A Simple Program.

simple.c:

/* CPE 101 */
/* Alex Dekhtyar */
/* A simple program */
/* This is a comment!
It ends here -> */

#include <stdio.h>
/* preprocessor directive */
#define KMPERMILE 1.6
/* preprocessor directive */

int main() { /* main() function: beginning */
    int miles; /* integer variable declaration */
    float km; /* float variable declaration */

    printf("\n\nMiles2Kilometers'R'US!\n\n"); /* output */
    printf("\n");
    printf("How many miles? ");

    scanf("%d", &miles); /* input */

    km = miles * KMPERMILE; /* assignment statement */

    printf("%d miles is %f kilometers\n", miles, km); /* formatted output */

    return 0; /* return statement */
}
/* end of main() function */
Anatomy

C Program. A C program consists of a sequence of comments, preprocessor directives, declarations and definitions.

simple.c program contains comments, preprocessor directives and a function definition (but no declarations. More on declarations - later in the course).

Comments. Comments are ignored by the C compiler.
Comments must be enclosed in /* and */.
A single comment can span multiple lines.

/* This is a comment in C */
/* This is a comment too */

Preprocessor directives. C comes with a preprocessor, a program that runs prior to submitting the C program to the compiler. C preprocessor plays a number of functions, but the most important ones are:

- Inclusion of multiple (library) files into the C program being compiled.
- Declaration of constants.

Preprocessor directives are instructions in the C source file that are read (parsed) and executed by the C preprocessor. We concentrate on two preprocessor directives now, will learn more about others later in the course:

- Inclusion of external C library files is achieved using the #include directive.
  Example. The #include preprocessor directive from simple.c brings in the external library of input-output functions. Both printf and scanf (see below) are functions from the stdio.h library.
- Constants are declared using #define preprocessor directive.

#include <stdio.h>
#define KMPERMILES 1.6

main() function definition. In order to compile into an executable file, a C program must contain the definition of the main() function. In any C program, the first statement to be executed is the first statement of its main() function. (i.e, execution starts at the beginning of the main() function).

In ANSI C (and we are studying the ANSI C standard) the definition of main() is as follows:

```c
int main() {
    <declarations>
    <statements>
    return 0;
}
```

Here, <declarations> is a sequence of variable declarations and <statements> is the sequence of steps (instructions) to be executed.
Variables. Variable is a named collection (sequence) of memory cells, which, at different times can contain different content (values). Variables have three key properties:

- **Name.** Variable name allows us to refer to the variable in the program (rather than referring to a specific memory address or addresses).
  It provides a level of abstraction between the code in a high-level language (like C) and the code in machine language.

- **Type.** The type of a variable is information about the class of values that can be stored in the variable. Type of a variable affects two things:
  1. the number of memory cells reserved for the variable, and
  2. the interpretation of the content of the variable.

- **Content.** The actual value(s) stored in the memory.

Variable Types. C has a wide range of types: from very simple to very complex. A table of some simple variable types is shown below.

<table>
<thead>
<tr>
<th>C type</th>
<th>Explanation</th>
<th>Range</th>
<th>number of bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer numbers</td>
<td>-32,767 ... 32,767</td>
<td>2</td>
</tr>
<tr>
<td>float</td>
<td>Floating point decimals</td>
<td>-3.438 ... 3.438</td>
<td>6 (7)</td>
</tr>
<tr>
<td>double</td>
<td>Double precision floating point decimals</td>
<td>-1.7308 ... 1.7308</td>
<td>10 (7)</td>
</tr>
<tr>
<td>char</td>
<td>a single character</td>
<td>-128 ... 127</td>
<td>1</td>
</tr>
<tr>
<td>bool</td>
<td>boolean value (true or false)</td>
<td>0, 1</td>
<td>1</td>
</tr>
<tr>
<td>void</td>
<td>no type</td>
<td>none</td>
<td>0</td>
</tr>
</tbody>
</table>

Variable declarations. A variable declaration has the syntax:

```
<type> <VariableName1>, <VariableName2>,...<VariableNameN>;
```

Here, `<type>` is one of C types (or one of user-created types — more about it at the end of the course), and `<VariableName1>, ...<VariableNameN>` are the names of the variable. A variable declaration can include one or more variable names. It must end with a semicolon (";”).

Variable Names Reserved Words and Identifiers. Variable names, dubbed user-defined identifiers are formed as follows:

- An identifier must consist of only letters (‘a’...’z’, ‘A’,...’Z’), digits (‘0’,..., ’9’) and underscores (‘_’).
- An identifier cannot begin with a digit.
- An identifier cannot be a C reserved word, nor can it be a C standard identifier.

Reserved Words. Some words (identifiers) are used in C language and have special meaning. They cannot be used as variable names, and their occurrence in C programs is interpreted according to special rules of C syntax.

Some of the reserved words in C are:

<table>
<thead>
<tr>
<th>type</th>
<th>reserved words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type names</td>
<td>int, float, double, bool, char, void</td>
</tr>
<tr>
<td>C control structures</td>
<td>if, while, for, return, case, switch, break, continue, goto, do</td>
</tr>
<tr>
<td>misc.</td>
<td>sizeof, typedef, union, struct</td>
</tr>
</tbody>
</table>
Standard Identifiers. Standard identifiers are the names of C functions from the standard C library. The complete list of these identifiers is found in Appendix B of your textbook.

Some standard identifiers you have already seen are printf and scanf.

I/O functions. C does not have statements for input and output. But it does have two standard library functions which are used for input and output.

printf(). printf() is the output function.
printf("string"); outputs the string. Strings must be enclosed in double quotes.
printf("formatted string", <variable1>,...,<variableN>); — formatted output. Format placeholders (“%d” for integer, ”%f” for floating point, ”%c” for character) are inserted into the the "formatted string". When output is generated, they are replaced with the values of variables in the variable list, in the order of appearance.

Example.
printf("%d miles is %f kilometers\n", miles, km);

Let’s assume that miles has value 1 and km has value 1.6. What the line above will produce as output is:

1 miles is 1.600000 kilometers

scanf(). scanf() is the input function.
scanf("format", &<variable1>,..., & <variableN>) reads from the standard input stream the values of variables whose names are <variable1>,...,<variableN>.

The format string consists of the format placeholders: "%d", "%f" (for floating point numbers), "%c".

Example
scanf("%d", &miles);

reads an integer value from the standard input (keyboard, by default) into the variable miles.

Note the use of the ampersand (&) in front of the variable name. While the real meaning of this symbol will be discussed at the end of the course, for now, it suffices to say that the presence of the & in front of the variable name, gives the scanf function the permission/ability to change the value of the variable.

Note: Both printf and scanf functions are part of the standard C library of functions. Standard C library consists of a number of header files, each containing the code for a set of "like-minded" functions.

All standard input/output functions reside in the stdio.h header file. This is why, any C program that contains input or output must contain the

#include <stdio.h>

preprocessor directive.

Constants

Constants are values used throughout the program. In C, each constant must come with a type, and each type has its own syntax for constant values.
Via the preprocessor `#define` directive, we can `name` some of the constants and use their names, rather than values in computations.

**Integer constants.** Integer constants can be either `signed` or `unsigned`. Examples of signed integer constants are:

\[-1 \quad +123 \quad -643054 \quad +23456\]

Examples of unsigned integer constants are:

\[0 \quad 1 \quad 999 \quad 34235 \quad 785\]

**Note:** Integer constants in C may **not** contain commas separating the orders. That is, 12,390 is **NOT** a correct integer constant.

**Floating point constants.** Floating point constants come in two different forms: the decimal point notation and the exponent notation and can be signed or unsigned. The decimal point constants have the form `IntegerPart.DecimalPart`. Examples of such constants are:

\[1.0 \quad 0.001 \quad -12.12 \quad 3.1415926 \quad +10000.00001 \quad .434\]

The constants in the exponent notation have the form

\[<Integer>.<Decimal>e<Exponent>\]

or

\[<Integer>.<Decimal>e-<Exponent>\]

The value of such a constant is `Integer.Decimal \times 10^{Exponent}` (or `Integer.Decimal \times 10^{-Exponent}`). The following are valid floating point constants:

\[2e-2 \quad 1.2e3 \quad 4.5e-5 \quad 6e+2\]

Their values are respectively:

\[2 \times 10^{-2} = 0.02; \quad 1.2 \times 10^{3} = 1200; \quad 4.5 \times 10^{-5} = 0.000045; \quad 6 \times 10^{2} = 600.\]

The following are **invalid** floating point constants

\[20 \quad /* \text{ no decimal point - it’s an integer constant */}\]
\[45e0.45 \quad /* \text{ the exponent must be an integer */}\]
\[3.342e \quad /* \text{ no exponent */}\]
\[344,453e200 \quad /* \text{ comma is not allowed */}\]

**Character constants.** Character constants are individual (ASCII) characters in single quotes (`'`). Examples of character constants are

\['a' \quad 'b' \quad 'C' \quad 'D' \quad 'T' \quad '3' \quad '\&' \quad ':' \quad ' ' \quad '_' \quad '='\]

The following are **NOT** valid character constants:

\"a" \quad /* \text{ this is a one-character string, but not a character */}\]
\"ab\" \quad /* \text{ only one character allowed inside single quotes */}\]
**String constants.** A string constant is any text inside **double quotation marks** ("). Examples are:

"a" "Hello, world!" "Train station" "LOL" "1234" "20,234" "/.?>$%$" "\n" "\t"

String constants can include special characters (escape sequences).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>new line</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\</td>
<td>\ (backslash)</td>
</tr>
<tr>
<td>%%</td>
<td>% (percent)</td>
</tr>
<tr>
<td>'</td>
<td>' (single quote)</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot; (double quote)</td>
</tr>
</tbody>
</table>