Chapter 2: Introduction to C Programming

A Simple C Program: Printing a Line of Text

* E.g.

```
/* A first program in C */
main()
{
    printf("Hello World.\n");
}
```

* The line(s) begin with "/*" and ends with "*/" indicating the line is a comment.
* Programmers insert comments to document programs and improve program readability.
* Comments do not cause the computer to perform any action when the program is run; i.e., comments are ignored by the C compiler and do not cause any machine language object code to be generated.

* The line "main()" is a part of every C program: every program in C begins executing at the function "main".
* The parentheses after "main" indicate that "main" is a program building block called a function. C programs contain one or more functions.

* The left brace, "{", must begin the body of every function (block).
* A corresponding right brace, "}", must end each function (block).
* This pair of braces and the portion of the program between the braces is called a block.

* The line "printf("Hello World.\n");" instructs the computer to perform an action, namely to print on the screen the string of characters marked by the quotation marks.
* The backslash(\) is called an escape character.
* When encountering a backslash, "printf" looks ahead at the next character and combines it with the backslash to form an escape sequence.
* The escape sequence "\n" means newline, and it causes the cursor to position to the beginning of the next line on the screen.

\n	Newline
\t	Tab
\r	Carriage return
\a	Alert
\\	Backslash
\"	Double quote

* The "printf" function is one of many functions provided in the C Standard Library.

* The entire line, including "printf", its arguments within the parentheses, and the semicolon (;), is called a statment.
* Every statement must end with a semicolon.

* Standard library functions (like "printf") are not part of the C programming language.
* Indent the entire body of each function (block) one level of indentation within the braces that define the body. This emphasizes the functional structure of programs and helps make programs easier to read.

* E.g.

```
main()
{
    printf("Hello ");
    printf("World.\n");
}
```

* E.g.

```
main()
{
    printf("Hello\nWorld.\n");
}
```

Another Simple C Program: Adding Two Integers

* E.g.

```
/* Addition program */
#include <stdio.h>

main()
{
    int integer1, integer2, sum;

    printf("Enter first integer\n");
    scanf("%d",&integer1);
    printf("Enter second integer\n");
    scanf("%d",&integer2);
    sum = integer1 + integer2;
    printf("Sum is %d\n", sum);

    return 0;
}
```

* The line "#include <stdio.h>" is a directive to the C preprocessor.
* This tells the preprocessor to include the contents of the standard input/output header file (stdio.h) in the program before it is compiled.
* The header file contains,
    1. information of the library functions
    2. declarations of the library functions
    3. information of the correctness of the functions called

* The line "int integer1, integer2, sum;" is a declaration.
* The letters "integer1", "integer2" and "sum" are the names of variables.
* A variable is a location in memory where a value can be stored for use by a program.
* These variables are of type "int" which means that these variables will hold integer values.
* All variables must be declared with a name and a data type immediately after the left brace that begins the body of a function (e.g. "main") before they can be used in a program.
* Several variables of the same type may be declared in one declaration, or we could have written three declarations, one for each variable.

* A variable name in C is any valid identifier.
* An identifier is a series of characters consisting of letters, digits, and underscores that does not begin with a digit.
* Only the first 31 characters of an identifier are required to be recognized by C compilers.

* C is case sensitive.
* Syntax error is caused when the compiler cannot recognize a statement; i.e. are violations of the language.
* Syntax error are also called compiler errors, or compiler-time errors.
* A logic error has it effect at execution time.
* A fatal logic error causes a program to fail and terminate prematurely.
* A nonfatal logic error allows a program to continue executing but to produce incorrect results.
* Some of the words in the C language (e.g. int, return and if) are keywords or reserved words of the language. The programmer must be careful not to use these words as identifiers such as variable names.

* The statement "scanf("%d", &integer1);" uses "scanf" to obtain a value from the user.
* The "scanf" function takes input from the standard input which is usually the keyboard.
* This "scanf" has two arguments:
    1. "%d" - the format control string: indicates the type of data that should be input by the user. (The letter "d" stands for decimal integer.)
    2. "&integer1" - the address/location of variable "integer1".
* When the computer executes "scanf", it waits for the user to enter a value for variable "integer1".

* The assignment statement "sum = integer1 + integer2;" calculates the sum of variables "integer1" and "integer2", and assigns the result to variable "sum" using the assignment operator "=".

* The statement "printf("Sum is %d\n",sum);" uses the "printf" function to print the literal "Sum is" followed by the numerical value of variable "sum" on the screen.
* This printf has two arguments:
    1. "Sum is %d\n" - the format control string: it contains some literal characters to be displayed, and it contains the conversion specifier "%d" indicatin that an integer will be printed.
    2. "sum" - the value to be printed.

* We could have combined the two statements:
```
        printf("Sum is %d\n", integer1 + integer2);
```

* The statement "return 0;" passes the value "0" back to the OS environment.

Memory Concepts
---
* Variable names corresponded to locations in the computer's memory.
* Every variable has a name, a type and a value.
* Whenever a value is placed in a memory location, the value overrides the previous value in that location.
* The values of variables may be used, but not destroyed, as the computer performed the calculation.

Arithmetic in C

Addition	+
Subtraction	-

```
    Multiplication  *
    Division        /
    Modulus         %
* Integer division yields an integer result.
* E.g. 7/4 evaluates to 1.
* Parentheses are used in much the same manner as in algebraic
expressions. E.g. a * (b + c)

* Operator precedence:
    1. ()
    2. *, /, or %
    3. + or -

Decision Making: Equality and Relational Operators
--------------------------------------------------
* Executable C statements either:
    1. perform action
    2. make decisions
* "if" control structure allows a program to make a decision based on
the truth of falsity of some statement of fact called a condition.
* If the condition is met (i.e., the condition is true) the statement
in the body of the "if" structure is executed.
* Whether the body statement is executed or not, after the "if"
structure completes, execution proceeds with the next statement after
the "if" structure.

* Conditions in "if" structures are formed by using the equality
operators and relational operators.
    Equality                    ==
    Inequality                  !=
    Greater than                >
    Less than                   <
    Greater than or equal to    >=
    Less than or equal to       <=
* In C, a condition may actually be any expression that generates a
zero (false) or nonzero (true) value.
* To aviod confusion, the equality operator is "double equals" and
the assignment operator is "single equal".
* E.g.
        /* Using if statements, relational
            operators, and equality operators */
        #include <stdio.h>

        main()
        {
            int num1, num2;

            printf("Enter two integers, and I will tell you\n");
            printf("the relationships they satisfy: ");
            scanf("%d%d", &num1, &num2);

            if (num1 == num2)
                printf("%d is equal to %d\n", num1, num2);

            if (num1 != num2)
                printf("%d is not equal to %d\n", num1, num2);

            if (num1 < num2)
                printf("%d is less than %d\n", num1, num2);
```

```
            if (num1 > num2)
                printf("%d is greater than %d\n", num1, num2);

            if (num1 <= num2)
                printf("%d is less than or equal to %d\n",
                                            num1, num2);

            if (num1 >= num2)
                printf("%d is greater than or equal to %d\n",
                                            num1, num2);

            return 0;
        }
* In C programs, white space characters such as tabs, newlines, and
spaces are normally ignored.
* However, it is not correct to split identifiers.
* Precedence and associativity of the operators
    ()              left to right
    * / %           left to right
    + -             left to right
    < <= > >=       left to right
    == !=           left to right
    =               right to left
```