Introduction to C Programming

Hello World

- Program to print a line of text on the screen

```
/* hello.c: Program to say hello to the world                  */
/*      Written by: Sanjiv K. Bhatia                            */
/*      Date:       September 5, 1996                           */

#include <stdio.h>

main()
{
    printf ( "Hello World !!!" );
}
```

  - Each line enclosed between `/*` and `*/` is a comment and is ignored by the compiler; its only use it to make the program more readable to humans and to record information (author, date, purpose)
  - The line with the keyword `include` is for the preprocessor to provide some information to the compiler, in this case the information about input/output (I/O) or standard I/O
  - The line `main()` is where the C program begins execution
    * This line is required in every C program
    * Name of the *function* or the purpose of the program
    * In the above program, the function `main` does not have any arguments or parameters
  - The braces { and } enclose the instructions to achieve the purpose of the program
    * This part of the program is called a *block*
  - The `printf` statement instructs the machine to write the message enclosed within parentheses on the screen
    * The string of characters within the double quotes are to be printed literally
    * The two characters `\n` denote *newline* and make the computer go to the next line after the message is printed
    * The backslash character \ is known as an escape character and causes the computer to perform a function depending on the character following it
  - The semicolon at the end of the `printf` command is a terminator for statements

Remarks on the program

- We'll assume that the program has been typed in a file named `hello.c`
- The extension `.c` is important and should not be missed
- The program will be compiled with the command

```
gcc -o hello hello.c
```

  - gcc is the name of the C compiler on the campus Unix machines
  - gcc is also the command to invoke the C compiler
  - `-o hello` tells the compiler that the machine language program shall be called `hello`, `-o` is the directive for *output* file name
  - Finally, `hello.c` is the source file

- Every program must have at least as much documentation as `hello.c`, purpose of the program, author, and date it was written
- The indentation is also very important as it aids in better reading of the program for humans
  - We'll establish a convention of using four spaces for indentation
- The same comment holds for white space

The second C program – Converting fahrenheit to celcius

```
/* f2c: Converting fahrenheit to celcius               */
/*   Written by: Sanjiv K. Bhatia                      */
/*   Date:       September 10, 1996                    */

#include <stdio.h>

main()
{
    int c,                          /* Temperature in celcius        */
        f;                          /* Temperature in fahrenheit     */

    f = 32;
    c = ( f - 32 ) * 100 / 180;

    printf ( "%d degree fahrenheit equal %d degree celcius\n", f, c );
}
```

- The line `int c, f` is a definition of two variables `c` and `f` and that they both are integers
- Note the assignment of a value to `f` and the calculation of `c` by applying the standard formula
- The `printf` statement contains two `%d` that are meant to hold the place for `f` and `c` during printing
- Change the value of `f` and see how it changes the value in the output

The second C program revised – Getting input from the user

```
/* f2ci: Converting fahrenheit to celcius                            */
/*   Written by: Sanjiv K. Bhatia                                    */
/*   Date:       September 10, 1996                                  */
/*   Revision 2: Added feature to get the input from user at run time */
/*   Date:       September 10, 1996                                  */

#include <stdio.h>

main()
{
    int c,                          /* Temperature in celcius        */
        f;                          /* Temperature in fahrenheit     */

    printf ( "Please enter the fahrenheits you want to be converted: " );
    scanf ( "%d", &f );
    c = ( f - 32 ) * 100 / 180;
```

```
    printf ( "%d degree fahrenheit equal %d degree celcius\n", f, c );
}
```

- The statement `scanf` gets input from the user
- `%d` indicates that the input will be read as an integer
- The `&` character is required to precede the name of the variable to be scanned in the `scanf` statement
- Notice the absence of `\n` in the first `printf` statement

Remarks, again

- Identifiers should be less than 31 characters long, with meaningful names
- As a convention, do not use upper case letters in the identifiers
- Words can be separated using an underscore to make them more readable

Memory concepts

- Correspondence between variable name and a memory location
- Non-destructive read out

Arithmetic in C

- Arithmetic symbols

Addition	`+`	$x + y$	`x + y`
Subtraction	`-`	$x - y$	`x - y`
Multiplication	`*`	$x \times y$	`x * y`
Division	`/`	$\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	`%`	x mod y	`x % y`

- Integer division vs. floating point division
- Division by zero error
- Rules of operator precedence
  - Expressions contained within parentheses are evaluated first of all and can be forced to change the order of evaluation of subexpressions
  - In the second step, the machine will evaluate multiplication, division, and modulus operations, from left to right within the expression
  - In the third step, the machine will evaluate addition and subtraction operators, again with left-to-right within the expression
  - Operator precedence table

`( )`
`* / %`
`+ -`

- Nested parentheses

Identifiers and keywords

- Identifiers
  - Names of variables and other program elements
  - Composed of
    * Letters – a, ..., z, A, ..., Z
    * Digits – 0, 1, ..., 9
    * Underscore – _
  - Identifiers must not begin with a digit
  - The capital and small letters are different
    * Typically, all identifiers to name variables or functions are constructed with lower case letters while the identifiers to denote constants and macros are constructed with all upper case letters
  - Identifiers are allowed to start with an underscore but generally, such identifiers are reserved for system use and should not be defined or declared in user programs
  - Examples of identifiers
    * `largest_number`
    * `a`
    * `A`
    * `H2O`
- Keywords
  - Identifiers that have been reserved for special purpose
  - Examples include `int`, `float`, `if`

Constants

- Integer constants
  - Sequence of digits, not beginning with 0, represents a decimal integer
  - Sequence of digits, beginning with 0, represents an octal integer and does not allow 8 or 9 in the string
  - Sequence of digits, beginning with `0x` (or `0X`), represents a hexadecimal integer, and allows the characters `a`, `b`, `c`, `d`, `e`, `f`, or their uppercase equivalents
  - Integer constants have type `int` by default
  - They assume the type `long int` (or `long`) if they are too large for `int` or if they are immediately followed by the letter `l` (or `L`), for example `12L`
- Floating-point constants
  - Always have type `double`
  - Contain a decimal point or the letter `E` (or `e`)
- Character constants
  - One character surrounded by single quotes
  - `'A'` or `'?'`

- Special characters and non-graphic characters
  - Denoted by a backslash \

`\n`	newline
`\t`	horizontal tab
`\b`	backspace
`\r`	carriage return
`\f`	form feed
`\\`	backslash
`\'`	single quote

  - Another form `\ddd` where each `d` is an octal digit
    - `ddd` specifies the desired internal value of a character
- String constants
  - Sequence of characters surrounded by double quotes
  - Backslash can be used for special characters
  - Double quotes within the string written as `\"`
  - `"ABC"` or `"%d\n\n"`
  - A null character (`\0`) is added immediately after the final character of a string
    - `"ABC"` is stored in four bytes as `"ABC\0"`
    - String constant `"A"` is different from character constant `'A'`
  - String constant `"ABC\nDEF"` is a two-line string
- Symbolic constants
  - Also known as "named constants"
  - Can be used to associated a meaningful name with any type of constant
  - Declared by using the preprocessor directive `#define`
    - `#define PI 3.1415926`
      associates the name `PI` with the value of π
  - The sign `#` should be the first non-whitespace character on the line
  - Comments are permitted on the preprocessor directive line
  - In general, there should be no semicolon at the end of the line; it may result in a compilation error later on in the program
  - It is common style to have all symbolic constant names in upper case
  - All the preprocessor directives can be expanded by using the option `-E` with the compile command as follows:

```
gcc -E foo.c
```

Data types

- Basic data types
  - `int` – Integer
    - One word in the machine (32 bits on most machines)
    - `short int` – Half word
    - `long int` – Double word
    - `short` and `long` are called qualifiers

- `float` – Floating point (single precision)
- `double` – Floating point (double precision)
  - `long double` is used for even greater precision
- `char` – Characters

Decision making with comparison and logical operators

- Used for making decisions about actions to be performed
- Selection, or `if` statement
  - Alters the control flow of a program depending upon the truth evaluation of a *condition*
- Conditions in the `if` statement are specified using the operators (known as equality and relational operators) in the following table

`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to
`==`	equal to
`!=`	not equal to
`&&`	logical and
`||`	logical or
`!`	not

- All the above operators associate from left to right, with relational operators at a higher precedence compared to the equality operators
- *The two-character operators should not be separated by a space character, neither their sequence can be reversed*
- *The equality operator (==) should not be confused with the assignment operator (=)*
- The `if` statement
  - Form 1: `if (expression) statement_1`
  - Form 2: `if (expression) statement_1 else statement_2`
  - Examples
    - `if ( x < 0 ) x = -x;`
    - `if ( u < 0 ) v = -u; else v = u;`
    - A preferable way to write the above will be

      ```
      if ( u < 0 )
          v = -u;
      else
          v = u;
      ```
- The expression `a < b` is of integer type
  - Evaluates to `0` if the expression is false
  - Evaluates to `1` if the expression is true
- A condition may be any expression that evaluates to an integer, with 0 being false and any nonzero value being true
- The expression `100 + ( 30 < 40 )` is valid and evaluates to `101`

- Expressions like

  ```
  if ( n != 0 ) ...
  ```

  can be replaced by

  ```
  if ( n ) ...
  ```

- Since false and true are coded by 0 and 1

`0 && ...`	evaluates to 0
`1 || ...`	evaluates to 1

- Improvement in program efficiency
  - Division by zero can be avoided by

    ```
    if ( n && q < k / n ) ...
    ```

- Importance of indentation and white space, again
- The `?:` operator
  - The above `if` statement could also be written as

    ```
    v = ( u < 0 ? -u : u );
    ```

- You can also execute more than one statement as the result of a success

```
if ( a >= b )
{
    p = 0;
    q = 1;
}
else
{
    s = 100;
    t = 200;
}
```

- Nesting is also allowed

```
if ( a >= b )
{
    x = 0;
    if ( a >= b + 1 )
    {
        xx = 0;
        yy = -1;
    }
    else
    {
        xx = 100;
        yy = 200;
    }
}
else
{
    x = 1;
    xx = -100;
    yy = 0;
}
```

- Cautions
  - If you omit braces, the `else` corresponds to the closest `if`
    - The statement

      ```
      if ( a == b ) if ( c == d ) x = 1; else x = 2;
      ```

      is equivalent to

      ```
      if ( a == b ) { if ( c == d ) x = 1; else x = 2;}
      ```

    - A better way to write the above is

      ```
      if ( a == b )
      {
          if ( c == d )
              x = 1;
          else
              x = 2;
      }
      ```

  - The brace that ends an `if` statement is not followed by a semicolon
  - Semicolons should not be regarded as statement separators; they belong to the statement itself