C Introduction Lesson Outline

hello_world.c

```
/*
 ************************************************
 *** Program: hello_world                     ***
 *** Author: Henry Neeman (hneeman@ou.edu)    ***
 *** Course: CS 1313 010 Spring 2024          ***
 *** Lab: Sec 014 Fridays 3:00pm              ***
 *** Description: Prints the sentence         ***
 ***   "Hello, world!" to standard output.    ***
 ************************************************
 */
#include <stdio.h>

int main ()
{ /* main */
    /*
     ********************************
     *** Execution Section (body) ***
     ********************************
     *
     * Print the sentence to standard output
     * (i.e., to the terminal screen).
     */
    printf("Hello, world!\n");
} /* main */
```

C Character Set

These are the characters that C recognizes (specifically, the characters that can be typed on a standard US English QWERTY computer keyboard):

- **<u>Letters</u>** (upper case and lower case)

```
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m
n o p q r s t u v w x y z
```

- **<u>Digits</u>**

```
0 1 2 3 4 5 6 7 8 9
```

- **<u>Special Characters</u>** (punctuation etc)

space (also known as ***<u>blank</u>***)

```
' " ( ) * + - / : =
! & $ ; < > % ? , .
^ # @ ~ ` { } [ ] \ |
```

C is Case Sensitive

C is ***case sensitive***: it distinguishes between **UPPER** case (CAPITAL) and **lower** case (small) letters.

Keywords in C – for example, the keyword **`int`** – **MUST** be in lower case. For example:

```
#include <stdio.h>

int main ()
{ /* main */
    int height_in_cm;

    height_in_cm = 160;
    printf("My height is %d cm.\n",
        height_in_cm);
} /* main */
```

Character String Literal Constant

A ***character string literal constant*** is a sequence of characters **delimited** by a double quote at the beginning and a double quote at the end.

A character string literal constant is also known as a ***character string literal*** or a ***string literal*** for short.

For example, in this **`printf` statement**:

```
printf("This is a printf.\n");
```

the following is a **string literal**:

```
"This is a printf.\n"
```

The **output** of this `printf` statement is:

```
This is a printf.
```

followed by a ***newline***, also known as a ***carriage return***.

String Literal Cannot Use Multiple Lines

A character string literal constant can only use one line; that is, both of its delimiters (the double quotes) **MUST** be on the same line of source code text.

So, this is **CORRECT**:

```
printf("This string literal takes one line");
printf(" and so does this string literal.\n");
```

And this is **WRONG WRONG WRONG**:

```
printf("This string literal takes
  more than one line so it's WRONG!\n");
```

Some compilers will accept this but won't be happy; other compilers will simply reject it.

Regardless, if this appears in a program in CS1313,

YOU WILL BE SEVERELY PENALIZED!

Multi-line String Literal Example

```
% cat bad_string_literal.c
#include <stdio.h>

int main ()
{ /* main */
    printf("This string literal takes
            more than one line so it's WRONG!\n");
} /* main */
% gcc -o bad_string_literal bad_string_literal.c
gcc bad_string_literal.c
bad_string_literal.c: In function ‘main’:
bad_string_literal.c:5: error: missing terminating " character
bad_string_literal.c:6: error: ‘more’ undeclared (first use in this function)
bad_string_literal.c:6: error: (Each undeclared identifier is reported only once
bad_string_literal.c:6: error: for each function it appears in.)
bad_string_literal.c:6: error: expected ‘)’ before ‘than’
bad_string_literal.c:6: error: missing terminating ' character
bad_string_literal.c:7: error: expected ‘;’ before ‘}’ token
```

Newline

In C, you can place a ***newline***, also known as a ***carriage return***, inside a string literal using:

```
\n
```

If a newline appears inside a string literal in the **source code**, then when the string literal is output, the newline causes the **output** to move to a new line.

Newline Example

```
% cat newline.c
#include <stdio.h>

int main ()
{ /* main */
    printf("Howdy do!\n");
    printf("This string literal contains a newline in the\nmiddle ");
    printf("but this string literal contains a newline at the end.\n");
    printf("So there!\n");
} /* main */
% gcc -o newline newline.c
% newline
Howdy do!
This string literal contains a newline in the
middle but this string literal contains a newline at the end.
So there!
```

Note: In general, it's better programming practice to **put newlines only at the end** of your string literals, **not in the middle**, because in the middle they can be difficult for programmers (for example, graders) to see.

White Space

White space is the general term for all of:

- blank spaces;
- tabs;
- carriage returns.

The term comes from the parts of standard typing paper that don't have any ink on them.

Statements

A ***statement*** in a program is like a sentence in a natural language: it's the smallest possible collection of words and punctuation that can stand by itself and have meaning.

For example:

```
printf("Hello, world.\n");
```

This statement is known as a `printf` statement (pronounced "print-eff").

It tells the compiler to output to the terminal screen the string literal

```
Hello, world.
```

followed by a newline.

Statement Terminator

In C, **every statement ends with a semicolon**,
which is known as the ***statement terminator***.

For example:

```
int height_in_cm;

height_in_cm = 160;
printf("My height is %d cm.\n",
    height_in_cm);
```

Notice: **A statement CAN take more than one line**
(but recall that **a string literal CAN'T take more than one line**).

The way you find the end of a statement is by
finding its statement terminator.

The way you find the start of a statement is by
finding the statement terminator of the previous statement.

Standard Input & Standard Output

- ***Standard input*** is when a user types at the keyboard. It is sometimes shortened to **`stdin`**, pronounced "standard in."
- ***Standard output*** is when the computer outputs to the terminal screen. It is sometimes shortened to **`stdout`**, pronounced "standard out."

In C:

- a `scanf` statement always inputs from `stdin`, and
- a `printf` statement always outputs to `stdout`.

More on this later.

Block Delimiters

The open curly brace, also known as the left brace,

{

acts as the start of a ***block*** and is known as the ***block open*.**

The close curly brace, also known as the right brace,

}

acts as the end of a ***block*** and is known as the ***block close*.**

The block open and block close are said to ***delimit*** the block: they indicate where the block begins and where the block ends.

Delimit: Indicate where something begins and ends.

What Is a Comment? #1

A ***comment*** is a piece of text in a source file that:

- **tells human beings** (for example, programmers) something useful about the program,

BUT

- is **ignored by the compiler**, so it has absolutely no affect on how the program runs.

In C, the start of a comment is indicated by

```
/*
```

and the end of a comment is indicated by

```
*/
```

All text appearing between these ***comment delimiters*** is part of the comment, and therefore is ignored by the compiler.

Delimit: Indicate where something begins and ends.

What Is a Comment? #2

A ***comment*** is a piece of text in a source file that:

- **tells human beings** (for example, programmers) something useful about the program,

BUT

- is **ignored by the compiler**, so it has absolutely no affect on how the program runs.

In C, the start of a comment is indicated by

```
/*
```

and the end of a comment is indicated by

```
*/
```

A comment **can use multiple lines** of text. The delimiters **DON'T** have to be on the same line.

Are Comments Necessary?

Comments are ignored by the compiler, so
strictly speaking they aren't needed to compile and run.

But, if you don't put them into one of your
CS1313 programming projects,
YOU MAY LOSE A FULL LETTER GRADE OR MORE
on that project.

Why?

Comments tell human beings useful things about your program.

They help **programmers** – including you, a month later when
you've forgotten everything about your program –
to understand your program.

They also tell **graders** that you know what the heck you're doing.

hello_world.c with Comments

```
/*
 ************************************************
 *** Program: hello_world                     ***
 *** Author: Henry Neeman (hneeman@ou.edu)    ***
 *** Course: CS 1313 010 Spring 2024          ***
 *** Lab: Sec 014 Fridays 3:00pm              ***
 *** Description: Prints the sentence         ***
 ***   "Hello, world!" to standard output.    ***
 ************************************************
 */
#include <stdio.h>

int main ()
{ /* main */
    /*
     ********************************
     *** Execution Section (body) ***
     ********************************
     *
     * Print the sentence to standard output
     * (i.e., to the terminal screen).
     */
    printf("Hello, world!\n");
} /* main */
```

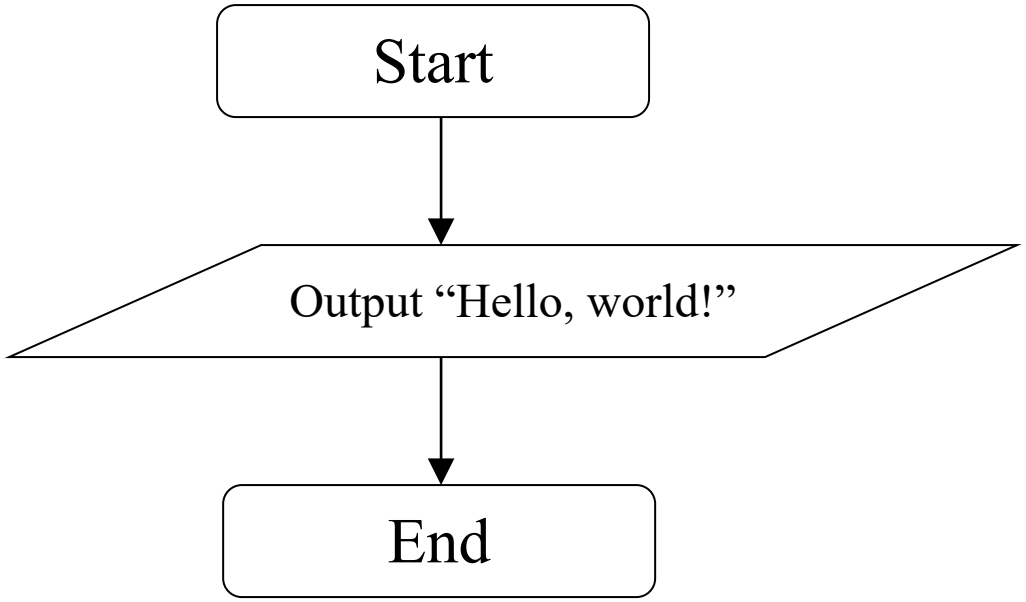
`hello_world.c` without Comments

```
#include <stdio.h>

int main ()
{
    printf("Hello, world!\n");
}
```

Flowchart for `hello_world.c`

```
int main ()
{
    printf("Hello, world!\n");
}
```



An **oval** denotes either the start or the end of the program, or a halt operation within the program (which we'll learn about later).

A **parallelogram** denotes either an input operation or an output operation.

An **arrow** denotes the flow of the program.

References:
http://www.edrawsoft.com/flowchart-symbols.php