Assertion in C

An assertion is a '*conditional check*' placed in a program to indicate that the developer thinks that the condtion has to be always true, at that place, for the program to work properly after that point. If the condition happens to be false, the program execution would be abruptly abhorted.

For example:

```
assert (y!=0); /* The developer thinks that y shouldn't be equal to 0. If not, the code would get abhorted */
z=x/y;
```

Whenever the program gets aborted because of an assertion, a corefile would be dumped. The developer can analyze the corefile (using any debugger, for example 'GDB') and pin-point the exact location of the failure, and then figure out why something unexpected happened. Thus assertions can be very useful for the developer during the early stages of development of a program.

Note that assertions would be enabled only during development and early stages of testing; they would be normally disabled during final testing. Assertions would always be disbaled when the final product is delivered to the customer.

How assertions are different from error handling

Assertions should be used to document logically impossible situations and discover programming errors— if the "impossible" occurs, then something fundamental is clearly wrong. This is distinct from error handling: most error conditions are possible, although some may be extremely unlikely to occur in practice. Using assertions as a general-purpose error handling mechanism is unwise: assertions do not allow for recovery from errors; an assertion failure will normally halt the program's execution abruptly. Assertions also do not display a user-friendly error message.

What is assert()?

Assert() is a macro defined in assert.h, as follows

```
void assert (int)
```

If the value (normally the result of the expression) passed is non-zero, nothing happens. However if the value happens to be zero, it prints an error message similar to the following one to STDERR and aborts the program

Assertion Failed: *file line :*

Normal uses

The simplest and most effective use of assertion is to specify and check input conditions to functions such as to check

- Pointers are not null
- Indexes and size values are non-negative and less than a known limit

Turning assertions off:

- By default,ANCI C compilers generate code to check assertions at run time.
- Assertion checking can be turned off by defining the NDEBUG flag to the compiler.
- This can be done either by inserting #define NDEBUG in a heder file such as stdhdr.h or while compiling use the option -dNDEBUG as

gcc -dNDEBUG filename

Example

```
#include <stdio.h>
#include <assert.h >
int main()
{
   int x,y,z;
   printf("Enter the 2 numbers to divide\n");
   scanf("%d %d",&x,&y);
   assert(y!=0);
   z=x/y;
   printf("x/y=%d\n",z);
   return;
}
```

Output:

```
Enter the 2 numbers to divide
3 0
Sample1: assert_test.c:8: main: Assertion `y!=0' failed.
Aborted
```

In this program if the user provides the value 0 for y, then assert() would get triggered and will abort the program.

Best Practice:

The following code

```
   assert(y!=0);
   z=x/y;
```

can be better written along with a string indicating what is the reason for the assertion, as

```
   assert(y!=0 && "y (denominator) cannot be zero");
   z=x/y;
```

Question: Whats wrong with this code

```
#include <stdio.h>
#include <assert.h>

void main () {
   char *my_str[]={"This", "is", "a", "string"};
```

```
    int index=0;
    while (index<4) {
        assert (++index && my_str[index-1]);
        printf("%s ", my_str[index-1]);
    }
    printf("\n");
}
```

Answer: When the code is compiled with assertions deisabled, index variable wouldn't get incremented and would result in an infinite loop. So never write any real code within an assert statement

Source:

http://datastructuresprogramming.blogspot.in/2010/05/assert-function.html