# Declaring constants

Data that will not change during the execution of a program should be stored in a constant container, rather than in a variable. This better enables the compiler to check the code for errors – if the program attempts to change the value stored in a constant the compiler will report an error and the compilation will fail.

A constant can be created for any data type by prefixing a variable declaration with the **const** keyword, followed by a space. Typically constant names appear in uppercase to distinguish them from (lowercase) variable names. Unlike variables, constants must always be initialized in the declaration. For example, the declaration of a constant for the math pi value looks like this:

```
const double PI = 3.1415926536 ;
```

The **enum** keyword provides a handy way to create a sequence of integer constants in a concise manner. Optionally, the declaration can include a name for the sequence after the **enum** keyword. The constant names follow as a comma-separated list within braces. For example, this declaration creates a sequence of constants:

```
enum suit { CLUBS , DIAMONDS , HEARTS , SPADES } ;
```

Each of the constants will, by default, have a value one greater than the preceding constant in the list. Unless specified the first constant will have a value of zero, the next a value of one, and so on. A constant can be assigned any integer value but the next constant in the list will always increment it by one.


The **typedef** keyword simply creates a nickname for a structure.

It is occasionally convenient to define a list of enumerated constants as a "custom data type" – by using the **typedef** keyword. This can begin the **enum** declaration and a chosen type name can be added at the end of the declaration. For example, this **typedef** statement creates a custom data type named "charge":

```
typedef enum { NEGATIVE , POSITIVE } charge ;
```

Variables can then be created of the custom data type in the usual way, which may legally be assigned any of the listed constants. Essentially these variables act just like an **int** variable – as they store the numerical integer value the assigned constant represents. For example, with the example above, assigning a **POSITIVE** constant to a **charge** variable actually assigns an integer of one.

1 Start a new program by specifying the C++ library classes to include and a namespace prefix to use

```
#include <iostream>
using namespace std ;
```


constant.cpp

2 Add a main function containing a final return statement

```
int main()
{
  // Program code goes here.
  return 0 ;
}
```

3 In the main function, insert statements to declare a constant and output using the constant value

```
const double PI = 3.1415926536 ;
cout << "6\" circle circumference: " << (PI * 6) << endl ;
```

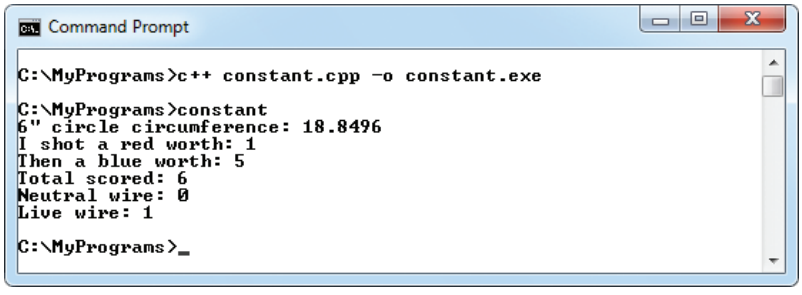
4 Next insert statements to declare an enumerated list of constants and output using some of those constant values

```
enum
{ RED=1, YELLOW, GREEN, BROWN, BLUE, PINK, BLACK } ;
cout << "I shot a red worth: " << RED << endl ;
cout << "Then a blue worth: " << BLUE << endl ;
cout << "Total scored: " << ( RED + BLUE ) << endl ;
```

5 Now insert statements to declare a custom data type and output its assigned values

```
typedef enum { NEGATIVE , POSITIVE } charge ;
charge neutral = NEGATIVE , live = POSITIVE ;
cout << "Neutral wire: " << neutral << endl ;
cout << "Live wire: " << live << endl ;
```

6 Save, compile, and run the program to see the output






**Hot tip**

In the PI declaration the * character is the C++ multiplication operator, and the backslash character in \" escapes the quote mark from recognition – so the string does not get terminated prematurely.