Symbolic constants in C

C provides three different mechanisms for defining names for numerical constants:

1. Preprocessor directive `#define` can create names for constants of any type;
2. Type qualifier `const` can define constant variables of any type; and,
3. Keyword `enum` can define constants of integer type.

Each of these three different mechanisms is better suited to certain situations than the others. The `#define` directive is the most powerful, and can be used in any situation where the other two might be used, but it is also the most dangerous as the preprocessor does not respect C syntax rules.

Variables qualified by `const` are generally preferred but, as `const`-qualified variables are not considered compile-time constants, they have one significant limitation; namely, a variable of type `const int` cannot be used to define the size of an array:

```c
#define ARRAYSIZE 10
int const ArraySize = 10;

double def_array[ARRAYSIZE];   // valid
double const_array[ArraySize]; // compile-time error
```

An enumeration constant does not suffer this limitation and, between them, `const` and `enum` can satisfy all symbolic constant operations for which a `#define` might be used. As `const` and `enum` are part of the C language proper, and abide by its rules, they are to be preferred over `#define` in general. For example,

```c
#define PI 3.14159         // not preferred
double const PI = 3.14159; // this is preferred over the define directive

#define ARRAYSIZE 10
enum { ARRAYSIZE = 10 };      // preferred over define directive
double enum_array[ARRAYSIZE]; // valid with either define or enum
```