Method 1: #define

Example

```c
int main(void)
{
    #define START_STATE 0
    #define POSSIBLE_COMMENT_STATE 1
    #define COMMENT_STATE 2
    ...
    int iState;
    ...
    iState = START_STATE;
    ...
    iState = COMMENT_STATE;
    ...
}
```

Notes

Good: Preprocessor does substitutions only for tokens.

```c
int iSTART_STATE; /* No substitution. */
```

Good: Preprocessor does not do substitutions within string literals.

```c
printf("What is the START_STATE?\n"); /* No substitution. */
```

Good: Works for any type of data.

```c
#define PI 3.14159
```

Bad: Handled by preprocessor; preprocessor does not respect scope.

Preprocessor replaces START_STATE with 0 from point of #define to end of file, not to end of function. Could affect subsequent functions unintentionally.

Bad: Handled by preprocessor; preprocessor does not respect context.

```c
int START_STATE;
```

After preprocessing, becomes:

```c
int 0; /* Compiletime error. */
```

Convention: Use all uppercase letters to reduce probability of unintended replacement.
**Method 2: Constant Variables**

**Example**

```c
int main(void)
{
    const int START_STATE = 0;
    const int POSSIBLE_COMMENT_STATE = 1;
    const int COMMENT_STATE = 2;
    ...
    ...
    int iState;
    ...
    iState = START_STATE;
    ...
    iState = COMMENT_STATE;
    ...
}
```

**Notes**

**Good:** Works for any type of data.

```c
    const double PI = 3.14159;
```

**Good:** Handled by compiler; compiler respects scope.

**Bad:** Does not work when specifying array lengths (unlike C++).

```c
    const int ARRAY_LENGTH = 10;
    ...
    int a[ARRAY_LENGTH]; /* Compiletime error */
```
Method 3: Enumerations

Example

```c
int main(void)
{
    /* Define a type named "enum State". */
    enum State {START_STATE, POSSIBLE_COMMENT_STATE, COMMENT_STATE, ...};

    /* Declare "eState" to be a variable of type "enum State".
    * 
    * Note: In C99, enum State eState;
    * 
    * eState = START_STATE;
    * 
    * eState = COMMENT_STATE;
    * 
    *
```

Notes

Interchangeable with type int.

```c
eState = 0;    /* Can assign int to enum. */
```

```c
i = START_STATE;    /* Can assign enum to int. START_STATE is an alias for 0, POSSIBLE_COMMENT_STATE is an alias for 1, etc. */
```

Good: Can explicitly specify values for names.

```c
enum State {START_STATE = 5,
            POSSIBLE_COMMENT_STATE = 3,
            COMMENT_STATE = 4,
            ...};
```

Good: Can omit type name, thus effectively giving symbolic names to int literals.

```c
enum {MAX_VALUE = 9999};
...
int i;
...
i = MAX_VALUE;
...
```

Good: Works when specifying array lengths.

```c
enum {ARRAY_LENGTH = 10};
...
int a[ARRAY_LENGTH];
...
```

Bad: Does not work for non-integral data types (e.g. type double).

```c
enum {PI = 3.14159};    /* Compiletime error */
```
Style Rules (see Kernighan and Pike Chapter 1)

(1) Use enumerations to give symbolic names to integral literals.

(2) Use const variables to give symbolic names to non-integral literals.

(3) Avoid using #define.