

Princeton University

COS 217: Introduction to Programming Systems

C Symbolic Constants

Method 1: #define

Example

```
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

Preprocessor does substitutions only for tokens.

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

Preprocessor does not do substitutions within string literals.

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

Bad: 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 unintentionally affect subsequent functions.

Bad: Preprocessor does not respect context.

```
int START_STATE;
```

After preprocessing, becomes:

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

Convention: Use all uppercase letters to reduce probability of unintended replacement.

Bad: gdb debugger prints value, not symbolic name.

Method 2: Constant Variables

Example

```
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: Compiler is allowed to not allocate storage if it need not.

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

```
const int ARRAY_LENGTH = 10;
...
int a[ARRAY_LENGTH]; /* Compiletime error */
```

Method 3: Enumerations

Example

```
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".
    enum State eState;
    ...
    eState = START_STATE;
    ...
    eState = COMMENT_STATE;
    ...
}
```

Notes

Mostly Bad: Interchangeable with type int.

```
eState = 0; /* Sadly, can mix enumerations and ints. */

i = START_STATE: /* START_STATE is an alias for 0, POSSIBLE_COMMENT_STATE
                  is an alias for 1, etc. */
```

Good: Can explicitly specify values for names.

```
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.

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

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

```
enum {PI = 3.14159}; /* Compiletime error */
```

Good: gdb debugger prints symbolic name, not value.

Style Rules (see Kernighan and Pike Chapter 1)

Use enumerations to give symbolic names to integral literals.

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

Avoid using #define to give symbolic names to literals.

Copyright © 2007 by Robert M. Dondero, Jr.