

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;
    ...
}
```

Strengths

Preprocessor does substitutions only for tokens.

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

Preprocessor does not do substitutions within string constants.

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

Simple textual substitution; works for any type of data.

```
#define PI 3.14159
```

Weaknesses

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.

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.

Convention: Place `#defines` at beginning of file, not within function definitions

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;
    ...
}
```

Strengths

Works for any type of data.

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

Handled by compiler; compiler respects context and scope.

Weaknesses

Does not work for 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

Interchangeable with type int.

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

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

Strengths

Can explicitly specify values for names.

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

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

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

Works when specifying array lengths.

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

Weakness

Does not work for non-integral data types.

```
enum {PI = 3.14159}; /* Compile-time 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**.

Copyright © 2009 by Robert M. Dondero, Jr.