Princeton University
COS 217: Introduction to Programming Systems
C Symbolic Constants

Approach 1: Macros

Example

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

Terminology

`START_STATE`, `POSSIBLE_COMMENT_STATE`, and `COMMENT_STATE` are *macros*.

Strengths

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. */
```

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 `#define`s at beginning of file, not within function definitions

Approach 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 aiNumbers[ARRAY_LENGTH];  /* Compile-time error */
```

Approach 3: Enumerations

Example

```
int main(void)
{
   enum State {START_STATE, POSSIBLE_COMMENT_STATE, COMMENT_STATE, ...};
   enum State eState;
   ...
   eState = START_STATE;
   ...
   eState = COMMENT_STATE;
   ...
}
```

Terminology

`enum State` is an *enumeration type.*

`START_STATE, POSSIBLE_COMMENT_STATE, …` are *enumeration constants*.

`eState` is an *enumeration*; it is of type `enum State`.

Notes

Enumeration constants are interchangeable with type int.

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

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

Strengths

Can explicitly specify values for enumeration constants.

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

Can define an *anonymous* enumeration type, 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 aiNumbers[ARRAY_LENGTH];
...
```

Weakness

Works for only literals of integral types (`char`, `short`, `int`, `long`, and unsigned variants thereof)

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

Style Rules (see Kernighan and Pike Chapter 1)

(1) Use **enumerations** to give symbolic names to literals of **integral** types (`char`, `short`, `int`, `long`, and unsigned variants thereof).

(2) Use **constant variables** to give symbolic names to literals of **non-integral** types (`float`, `double`, `long double`, and string).

(3) Avoid using **`macros`** to give symbolic names to literals.