Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Spring 2023

C Preprocessor

Preprocessor

- The C language uses the preprocessor to expand its power and notation
- Preprocessor directives
  - Lines beginning with a #
  - Communicates with the preprocessor
  - `#include`
  - `#define`
  - `#undef`
  - `#if`
  - `#endif`
  - ...

Use of `#include`

- Standard header files
  - `#include <stdio.h>`
  - `#include <stdlib.h>`
  - The preprocessor looks for the file only in the places where standard header files are stored (e.g., `/usr/include` etc.), not in the current directory

- User header files
  - `#include "myheader.h"`
  - Search is made first in the current directory
  - And then in other system-dependent places

Use of `#define` (1)

- `#define` *identifier token_string*
  - The preprocessor replaces every occurrences of *identifier* by *token_string* in the remainder of the file, except in quoted string
  - *token_string* is optional
- The use of simple `#define` can improve
  - Program clarity
  - Program portability

```c
#define SECONDS_PER_DAY     (60*60*24)
#define PI        3.14159
#define C         299792.458      /* speed of light in km/sec */
#define EOF       (-1)            /* typical end-of-file value */
#define MAXINT    2147483647      /* largest 4-byte integer */
```

Use of `#define` (2)

- Alter the syntax of C toward users' preference

```
#define EQ      ==
#define do              /* blank */


while (i EQ 1) do {                         while (i == 1) {
    ...                  <=>                    ...
}                                           }
```

Macros with Argument (1)

- `#define` can be used to write macro definitions with parameters
  - `#define` *identifier*(*identifier*, …, *identifier*) *token_string*$_{opt}$

  - `#define SQ(x)    ((x) * (x))`
```
SQ(7 + w)       ➔  ((7 + w) * (7 + w))
SQ(SQ(*p))      ➔ ((SQ(*p)) * (SQ(*p)))
                ➔ ((((*p) * (*p))) * (((*p) * (*p))))
```

  - `#define SQ(x)    x * x`
```
SQ(a + b)       ➔  a + b * a + b  ≠  ((a + b) * (a + b))
```

  - `#define SQ(x)    (x) * (x)`
```
4 / SQ(2)       ➔  4 / (2) * (2)  ≠  4 / ((2) * (2))
```

Macros with Argument (2)

- Erroneous `#define`

  - `#define SQ (x)   ((x) * (x))`

    ```
    SQ(7)          ➔  (x) ((x) * (x))(7)
    ```

  - `#define SQ(x)    ((x) * (x));`

    ```
    if (x == 2)
        x = SQ(y);  ➔       x = ((y) * (y));;          /* a common error */
    else
        x++;
    ```

Macros with Arguments (3)

- Macros are frequently used to replace function calls by inline code
  - `#define min(x, y)  (((x) < (y))? (x) : (y))`

    `m = min(u, v);` ➔ `m = (((u) < (v))? (u) : (v));`
  - `#define min4(a,b,c,d)  min(min(a,b), min(c,d))`

- A macro definition can use both functions and macros in its body
  - `#define SQ(x)      ((x) * (x))`
  - `#define CUBE(x)    (SQ(x) * (x))`
  - `#define F_POW(x)   sqrt(sqrt(CUBE(x)))`   */* fractional power: 3/4 */*

Macros in `stdio.h` and `ctype.h`

- `<stdio.h>`

```
#define getchar      getc(stdin)
#define putchar(c)   putc(c, stdout)
```

- `<ctype.h>`
  - `c` is a variable of integral type, such as `char` or `int`
  - The value of `c` stored in memory does not get changed

Macro	Return value
`isalnum(c)`	true if c is a letter or digit
`isxdigit(c)`	true if c is a hexadecimal digit
`isspace(c)`	true if c is a white space character
`ispunct(c)`	true if c is a punctuation character
`isalpha(c)/isdigit(c)`	true if c is a letter/digit
`isupper(c)/islower(c)`	true if c is an uppercase/lowercase letter
`isprint(c)/iscntrl(c)`	true if c is a printable/control character
`toupper(c)`	corresponding uppercase value or c
`tolower(c)`	corresponding uppercase value or c
`toascii(c)`	corresponding ASCII value

Conditional Compilation

- `#if` *expression*
  - The *expression* consists of constants, arithmetic/logical operators, macros, `defined()` operator, etc.
  - The conditional succeeds if the value of *expression* is nonzero
- `#ifdef` *macro*
  - The conditional succeeds if *macro* is defined (by `#define` or `gcc -Dmacro`)
- `#ifndef` *macro*
  - The conditional succeeds if *macro* is NOT defined
- `#else, #elif, #endif`
  - `#endif` always matches the nearest `#ifdef`, `#ifndef`, or `#if`
- `#undef` *identifier*
  - Removes the current definition of *identifier*

Conditional Compilation: Examples

```
#define DEBUG
#ifdef DEBUG
    printf("debug: a = %d\n", a);
#endif
```

```
#define DEBUG
#if defined(DEBUG)
    printf("debug: a = %d\n", a);
#endif
```

```
#define DEBUG   1
#if DEBUG
    printf("debug: a = %d\n", a);
#endif
```

```
#include "everything.h"
#undef PIE
#define PIE "I like apple."
```

```
#if defined(HP9000) || defined(SUN4) && !defined(VAX)

    ...         /* machine-dependent code */

#endif
```

Predefined Macros

`__DATE__`	: a string containing the current date
`__TIME__`	: a string containing the current time
`__STDC__`	: if the implementation follows ANSI C Standard, the value is a nonzero integer
`__FILE__`	: the source file name (string) containing this macro
`__LINE__`	: an integer representing the current line number

Stringization

- Preprocessing operator #
  - Convert a macro argument into a string constant

```c
#include <stdio.h>

#define print_var(x)    printf(#x " is %d\n", x)

void main(void)
{
    int a = 1, b = 2;

    print_var(a);
    print_var(b);
}
```

$\Rightarrow$

```c
void main(void)
{
    int a = 1, b = 2;

    printf("a" " is %d\n", a);
    printf("b" " is %d\n", b);
}
```

Concatenation

- Preprocessing operator ##
  - Merge two tokens into one while expanding macros

```
struct command {
    char *name;
    void (*function)(void);
};

#define COMMAND(name) { #name, name ## _command }

struct command cmds[] =
{
    COMMAND(quit),
    COMMAND(help),
    /* ... */
};
```

```
struct command {
    char *name;
    void (*function)(void);
};


struct command cmds[] =
{
    { "quit", quit_command },
    { "help", help_command },
    /* ... */
};
```

Macro assert()

```
#define assert(expr)                          \
    if (!(expr)) {                            \
        printf("\n%s%s%s%s%s%d\n",            \
            "Assertion failed: ", #expr,      \
            " in file ", __FILE__,            \
            " at line ", __LINE__);           \
        abort();                              \
    }

void main(void) {
    int n = 10;
    assert(n > 0 && n < 7);
}
```

```
void main(void) {
    int n = 10;

    if (!(n > 0 && n < 7)) { printf("\n%s%s%s%s%s%d\n", "Assertion failed: ",
    "n > 0 && n < 7", " in file ", "assert.c", " at line ", 12); abort(); };
}
```

qsort()

-
  ```
  qsort(void *base, size_t nmemb, size_t size,
            int (*compare)(const void *, const void*));
  ```
  - Sorts an array with `nmemb` elements of size `size`
  - The `base` argument points to the start of the array

- The comparison function `compare()` returns *x* where
  - *x* < 0: if the first argument is less than the second
  - *x* == 0: if the first argument is equal to the second
  - *x* > 0: if the first argument is greater than the second
  - If two members compare as equal, their order in the sorted array is undefined

Example: Quicksort (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N   11          /* size of the array */

int cmp(const void *vp, const void *vq) {
    const double *p = vp, *q = vq;
    double diff = *p - *q;

    return (diff == 0)? 0 : ((diff < 0)? -1 : 1);
}

void fill_array(double *a, int n) {
    int i;

    srand(time(NULL));
    for (i = 0; i < n; i++)
        a[i] = (rand() % 1000) / 10.0;
}
```

Example: Quicksort (2)

```c
void prn_array(char *msg, double *a, int n) {
    int i;

    printf("---\n%s sorting", msg);
    for (i = 0; i < n; i++) {
        if (i % 6 == 0)
            putchar('\n');
        printf("%10.1f", a[i]);
    }
    putchar('\n');
}

int main(void) {
    double a[N];

    fill_array(a, N);
    prn_array("before", a, N);
    qsort(a, N, sizeof(double), cmp);
    prn_array("after", a, N);
    return 0;
}
```

```
$ ./a.out
---
before sorting
      18.1      78.5       9.3       8.0      75.9       4.4
       1.0      43.6      23.5      93.5      38.6
---
after sorting
       1.0       4.4       8.0       9.3      18.1      23.5
      38.6      43.6      75.9      78.5      93.5
```

Example: Generic Quicksort (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define N       11              /* size of the array */
#define frac(x)                 ((x) - (int)(x))
#define random_char()           (rand() % 26 + 'a')
#define random_float()          (rand() % 1000 / 10.0)

#define FILL(array, sz, type)   if (strcmp(type, "char") == 0)     \
                                    for (i = 0; i < sz; i++)        \
                                        array[i] = random_char();   \
                                else                                \
                                    for (i = 0; i < sz; i++)        \
                                        array[i] = random_float()

#define PRINT(array, sz, fstr)  for (i = 0; i < sz; i++)            \
                                    printf(fstr, array[i]);         \
                                putchar('\n')
```

Example: Generic Quicksort (2)

```c
int cmp_float(const void *vp, const void *vq) {
    const float *p = vp, *q = vq;
    float x;

    x = frac(*p) - frac(*q);
    return ((x < 0.0)? -1 : (x == 0.0)? 0 : 1);
}

int cmp_char(const void *vp, const void *vq) {
    const char *p = vp, *q = vq;
    return (*p - *q);
}
```

```
b g p y d c e e p m q
b c d e e g m p p q y
---
8.0     17.7    15.7    92.1    88.7    8.2     39.9
20.8    85.3    11.0    26.1
8.0     11.0    92.1    26.1    8.2     85.3    88.7
15.7    17.7    20.8    39.9
```

```c
int main(void)
{
    char a[N];
    float b[N];
    int i;

    srand(time(NULL));
    FILL(a, N, "char");
    PRINT(a, N, "%-2c");
    qsort(a, N, sizeof(char), cmp_char);
    PRINT(a, N, "%-2c");
    printf("---\n");
    FILL(b, N,  "float");
    PRINT(b, N, "%-8.1f");
    qsort(b, N, sizeof(float), cmp_float);
    PRINT(b, N, "%-8.1f");
    return 0;
}
```

Example: Generic Quicksort (3)

```c
#define FILL_FUNC(type)         random_ ## type()
#define CMP_FUNC(type)          cmp_ ## type

#define FILL(array, sz, type)                    \
            for (i = 0; i < sz; i++)             \
                array[i] = FILL_FUNC(type)

#define QSORT(array, sz, type)                   \
            qsort(array, sz, sizeof(type), CMP_FUNC(type))
```

```
l l y n e c b z i r e
b c e e i l l n r y z
---
44.4    97.9    20.8    68.1    93.7    6.7     8.9
86.1    96.5    31.7    45.3
68.1    86.1    45.3    44.4    96.5    93.7    6.7
31.7    20.8    8.9     97.9
```

```c
int main(void)
{
    char a[N];
    float b[N];
    int i;

    srand(time(NULL));
    FILL(a, N, char);
    PRINT(a, N, "%-2c");
    QSORT(a, N, char);
    PRINT(a, N, "%-2c");
    printf("---\n");
    FILL(b, N,  float);
    PRINT(b, N, "%-8.1f");
    QSORT(b, N, float);
    PRINT(b, N, "%-8.1f");
    return 0;
}
```