C Preprocessor

Defining Macros, Include Guards, Conditional Compilation

SOFTWARE UNIVERSITY FOUNDATION

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

1. The C Preprocessor
2. Using **`#include`**
3. Symbolic Constants with **`#define`**
4. Macros with **`#define`**
5. Conditional Compilation
6. Include guards and **`#pragma`**

The C Preprocessor

- The preprocessor is executed before a program is compiled
  - Includes code of referenced libraries
  - Performs replacement of symbolic constants and macros
  - Provides conditional code compilation
  - Conditional execution of preprocessor directives
  - …and more

Using the #include directive

- The **#include** directive causes a copy of the specified file to be included in the place of the directive:

```
#include <stdlib.h>
#include "custom-list"
```

  - If the name is enclosed in `""`, the preprocessor starts the search in the current directory
  - If the name is enclosed in `<>`, the preprocessors searches system directories

Using the #define directive

- The **#define** directive creates symbolic constants

```
#define <identifier> <replacement-text>
```

  - Replaces all occurrences of **<identifier>** with replacement
  - Example:

```
#define SIZE 10
…
char text[SIZE];
text[SIZE – 1] = '\0';
strncpy(text, "Hello", SIZE – 1);
```

⇨

```
#define MAX_LENGTH 10
…
char text[10];
text[10 – 1] = '\0';
strncpy(text, "Hello", 10 – 1);
```

Macros

- A macro is a symbolic fragment of code
  - Can take arguments just like a function

```c
#define PI 3.14159
#define CIRCLE_AREA(r) ((PI) * (r) * (r))
...
double area = CIRCLE_AREA(5);
printf("area: %f\n", area);
```

Replaces the reference with the macro text

⇩

```c
double area = ((3.14159) * (5) * (5));
printf("area: %f\n", area);
```

Multiline Macros

- The preprocessor supports multiline macros
  - Lines should end with \ character to denote the macro continues

```
#define DEBUG 1
#define debug_print(...) \
        do \
        {  \
            if (DEBUG) \
                fprintf(stderr, ##__VA_ARGS__); \
        } while (0)
```

  - The **do-while(0)** loop prevents replacement side effects
    - More: http://stackoverflow.com/a/1644898

Multiline Macros

Side Effects

Macros vs Functions

- Using macros instead of functions is dangerous
  - There is no type-checking
  - They often produce unexpected side effects
  - Difficult to debug
- In the past they were used to reduce function call overhead
  - Nowadays inline functions are supported (since C99)
    - The compiler injects the function instructions in the calling function
    - Note: `inline` is only a hint to the compiler, it may be ignored

Macros

Live Demo

Conditional Compilation

- The **#if** and **#endif** directive allows C to compile certain parts of a program only if a condition is true

```
#define DEBUG 1

int main()
{
    char *fileName = "file.txt";
    FILE* file = fopen(fileName, "r");
#if DEBUG
    printf("Opened file %s\n", fileName);
#endif
    ...
```

Compiles code if **true**

Conditional Definition Execution – Example

```
#define DEBUG 1

#if DEBUG
#define log_err(fmt, ...) \
    fprintf(stdout, fmt, __VA_ARGS__);
#else
#define log_err(M, ...)
#endif
```

Defines a macro if **DEBUG** is set to **1**

Defines a macro without a body otherwise

Using the Debug Macro – Example

```
int main() {
    char *fileName = "file.txt";
    FILE* file = fopen(fileName, "r");
    if (!file)
    {
        perror(NULL);
        exit(1);
    }

    debug_print("Opened file %s\n", fileName);
    fclose(file);
    debug_print("Closed file %s\n", fileName);

    return 0;
}
```

Debug Print Macro

Live Demo

Standard Predefined Macros

- The C preprocessor has several predefined macros
  - `__LINE__` – expands to the current input line number
  - `__FILE__` – expands to the name of the current input file
  - `__TIME__` – expands to the time preprocessor is run (e.g. 23:03:00)
  - More: https://gcc.gnu.org/onlinedocs/cpp/Standard-Predefined-Macros.html
    - `__FILE__` and `__LINE__` are useful for error reporting

Error Print Macro

- Printing formatted message to the standard error stream

```
#define clean_errno() (errno == 0 ? "None" : strerror(errno))

#define log_err(M, ...) \
    fprintf(stderr, "[ERROR] (%s:%d: errno: %s) " M "\n", \
        __FILE__, __LINE__, clean_errno(), ##__VA_ARGS__)
```

  - Accepts message `M` and argument list `...`
  - Concatenates `"[Error] …"` with `M` and `"\n"`
  - `##__VA_ARGS__` refers to `...`

Using the Error Print Macro

```
int main()
{
    long mem = 1l << 32;
    char *buffer = malloc(mem);
    if (!buffer)
    {
        log_err("Unable to alloc %ld bytes for buffer", mem);
        exit(1);
    }

    return 0;
}
```

Passing a non-literal will not compile

```
Output:
[ERROR] (main.c:38: errno: Cannot allocate
memory) Need 4294967296 bytes for buffer
```

Error Print Macro

Live Demo

Include Guards

- Include guards help avoid double inclusion of header files
  - The following code produces a compilation error:

main.c

```c
#include "geometry.h"
#include "math.h"

int main()
{
   ...
```

geometry.h

```c
#include "math.h"
typedef struct Triangle
{
    Segment segmentA;
    Segment segmentB;
    Segment segmentC;
} Triangle;
```

math.h

```c
typedef struct Segment
{
    float a;
    float b;
} Segment;
```

  - The contents of `math.h` get included twice in `main.c`

Using Include Guards

- Code is put inside a conditional compilation construct
  - Guaranteed to be included only once inside a compilation unit

geometry.h

```
#include "math.h"
#ifndef _GEOMETRY_H
#define _GEOMETRY_H

typedef struct Triangle {
    Segment segmentA;
    Segment segmentB;
    Segment segmentC;
} Triangle;

#endif
```

✓

math.h

```
#ifndef _MATH_H
#define _MATH_H

typedef struct Segment {
    float a;
    float b;
} Segment;

#endif
```

✓

#pragma once

- **#pragma once** is a non-standard preprocessor directive
  - Acts like an include guard, but is shorter
  - Supported on most major compilers (GCC, Clang, MSVC)

```
#pragma once

typedef struct Segment {
    float a;
    float b;
} Segment;
```

Same as enclosing code in include guard

C Programming – Preprocessor

SOFTWARE UNIVERSITY FOUNDATION

xssoftware

SmartIT

Komfo
SOCIAL MEDIA MARKETING AT SCALE

NETPEAK
интернет-маркетинг для бизнеса

Questions?

LUXOFT

SUPERHOSTING®.BG

SoftwareGroup
doing it right

INDEAVR
Serving the high achievers

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

https://softuni.bg/trainings/1212/C-Programming-October-2015

License

Free Trainings @ Software University

- Software University Foundation – softuni.org
- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg