The Preprocessor (1)

Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

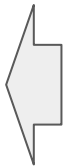
Outline

- Introduction - Directives
- Introduction - Preprocessor
- How the Preprocessor Works
- Preprocessing Directives
- Preprocessing Directives - General Rules

Introduction - Directives

- `black` words: appear in the C program exactly as shown
- *`blue`* words: text need to be written by the programmers

```
directives

int main(void)
{
   statements
}
```



ccu.c

```
#include <stdio.h>

int main(void)
{
  printf("Hello CCU.\n");
  return 0;
}
```

C Fundamentals - how to execute / run?

- Store this program in a file named `ccu.c` (or any other name you like)
- The file name doesn't matter, but the `.c` extension is often required.
- Before a program can be executed, there are three steps.

- Run the commands that begin with #
- For example,

```
#include
<stdio.h>
```

Compiler then translates the program into machine instructions (*object code; binaries*)

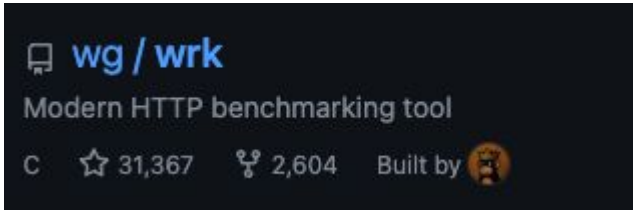
linker combines the object code with any additional code needed to generate a complete **executable** program (for example, .exe).

Introduction - Preprocessor

- Directives such as `#define` and `#include` are handled by the ***preprocessor,*** a piece of software that edits C programs just prior to compilation.
- Its reliance on a preprocessor makes C (along with C++) unique among major programming languages.
- The preprocessor is a powerful tool, but it also can be a source of hard-to-find bugs.

Introduction - Preprocessor

- Directives are widely used by professional programmers!
- Let’s see some examples.
- Trending C-based project on Github
  - https://github.com/wg/wrk
  - https://github.com/wg/wrk/blob/master/src/ae.c



```
43  #include "ae.h"
44  #include "zmalloc.h"
45  #include "config.h"
46
47  /* Include the best multiplexing layer supported by this system.
48   * The following should be ordered by performances, descending. */
49  #ifdef HAVE_EVPORT
50  #include "ae_evport.c"
51  #else
52      #ifdef HAVE_EPOLL
53      #include "ae_epoll.c"
54      #else
55          #ifdef HAVE_KQUEUE
56          #include "ae_kqueue.c"
57          #else
58          #include "ae_select.c"
59          #endif
60      #endif
61  #endif
```

How the Preprocessor Works

- The preprocessor looks for ***preprocessing directives,*** which begin with a `#` character.
- We've encountered the `#define` and `#include` directives before.
- `#define` defines a ***macro***—a name that represents something else, such as a **constant**.

```
#include <stdio.h>
#define DAYS_OF_YEAR 365

int main(void){
  int day_of_two_years = 2 * DAYS_OF_YEAR;
  return 0;
}
```

How the Preprocessor Works

- The preprocessor responds to a `#define` directive by **storing** the name of the macro along with its definition.
- When the macro is used later, the preprocessor "**expands**" the macro, replacing it by its defined value.

```
#include <stdio.h>
#define DAYS_OF_YEAR 365

int main(void){
  int day_of_two_years = 2 *  365;
  return 0;
}
```

How the Preprocessor Works

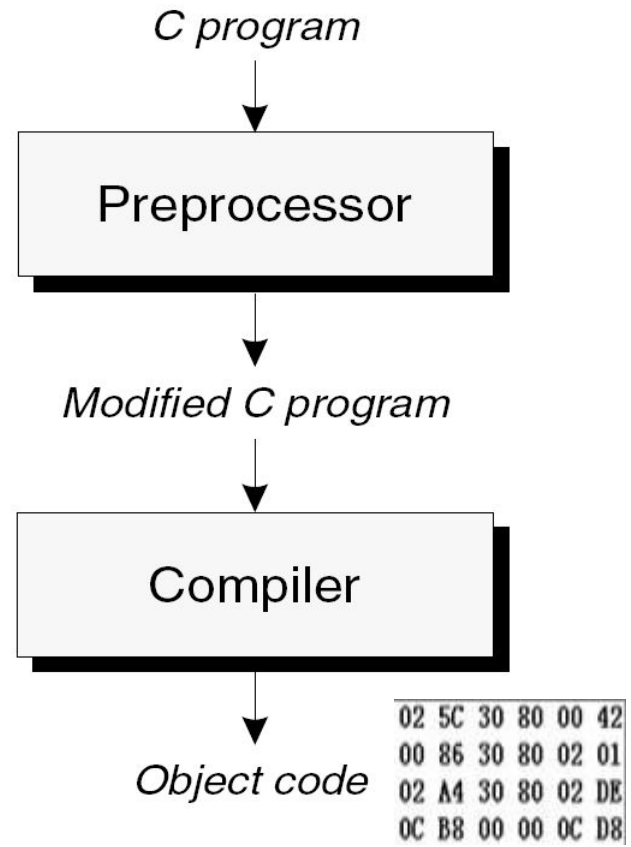
- `#include` tells the preprocessor to open a particular file and "include" its contents as part of the file being compiled.
- `#include <stdio.h>`
  - instructs the preprocessor to open the file named `stdio.h` and bring its contents into the program.

```
#include <stdio.h>

int main(void){
     …
     return 0;
}
```

```
…      codes from stdio.h
…

int main(void){
     …
     return 0;
}
```

How the Preprocessor Works

- The preprocessor's role in the compilation process:
- The input to the preprocessor is a C program, possibly containing directives.
- The preprocessor executes these directives, removing them in the process.
- The preprocessor's output goes directly into the compiler.



How the Preprocessor Works

```
#include <stdio.h>

#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)

int main(void)
{
  float f, c;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &f);

  c = (f - FREEZING_PT) * SCALE_FACTOR;
  printf("Celsius equivalent is: %.1f\n",
c);

  return 0;
}
```

```
Lines brought in from stdio.h

Blank line
Blank line

int main(void)
{
  float f, c;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &f);

  c = (f - 32.0f) * (5.0f / 9.0f);
  printf("Celsius equivalent is: %.1f\n",
c);

  return 0;
}
```

How the Preprocessor Works

- In the early days of C, the preprocessor was a separate program.
- Nowadays, the preprocessor is often part of the compiler, and some of its output may not necessarily be C code.
- Most C compilers provide a way to view the output of the preprocessor.
  - For example, GCC will do so when the `-E` option is used
  - GCC: is one of the most popular compiler of C

Preprocessing Directives

- Most preprocessing directives fall into one of three categories:
- ***Macro definition.*** The `#define` directive defines a macro; the `#undef` directive removes a macro definition.
- ***File inclusion.*** The `#include` directive causes the contents of a specified file to be included in a program.

```
#include <stdio.h>
#define DAYS_OF_YEAR 365

int main(void){
  int day_of_two_years = 2 * DAYS_OF_YEAR;
  return 0;
}
```

Preprocessing Directives

- Most preprocessing directives fall into one of three categories:
- ***Conditional compilation.*** The `#if`, `#ifdef`, `#ifndef`, `#elif`, `#else`, and `#endif` directives allow blocks of text to be either included in or excluded from a program.

```
#ifndef BUFFER_SIZE
#define BUFFER_SIZE 256
#endif
…
```

Preprocessing Directives - General Rules

- Several rules apply to all directives.
- ***Directives always begin with the # symbol.***
  - The # symbol need not be at the beginning of a line, as long as only white space precedes it.

```
#include <stdio.h>
include <stdio.h> // Wrong!
```

Preprocessing Directives - General Rules

- ***Any number of spaces and horizontal tab characters may separate the tokens in a directive.***
- Example:

```
#    define     N     100
# include    <stdio.h>
# include    <  stdio.h  > //Wrong! <stdio.h> is a token
```

Preprocessing Directives - General Rules

- ***Directives always end at the first new-line character, unless explicitly continued.***
- To continue a directive to the next line, end the current line with a \ character

```
9  #include <stdio.h>
10 #define TEST (1 + \
11                2 * \
12                10)
13 int main()
14 {
15     printf("%d", TEST);
16
17     return 0;
18 }
```

```
21


...Program finished with exit code 0
Press ENTER to exit console.
```

Preprocessing Directives - General Rules

- ***Directives can appear anywhere in a program.***
  - Although `#define` and `#include` directives usually appear at the beginning of a file, other directives are more likely to show up later.
- ***Comments may appear on the same line as a directive.***
  - It's good practice to put a comment at the end of a macro definition:

```
#include <string.h> //include string lib
#define FREEZING_PT 32.0f // freezing point of water
```

Summary

- Introduction - Directives
- Introduction - Preprocessor
- How the Preprocessor Works
- Preprocessing Directives
  - What are the three categories of directives
- Preprocessing Directives - General Rules
  - What are the general rules of directives