The PreProcessor

CS 580U - Fall 2017

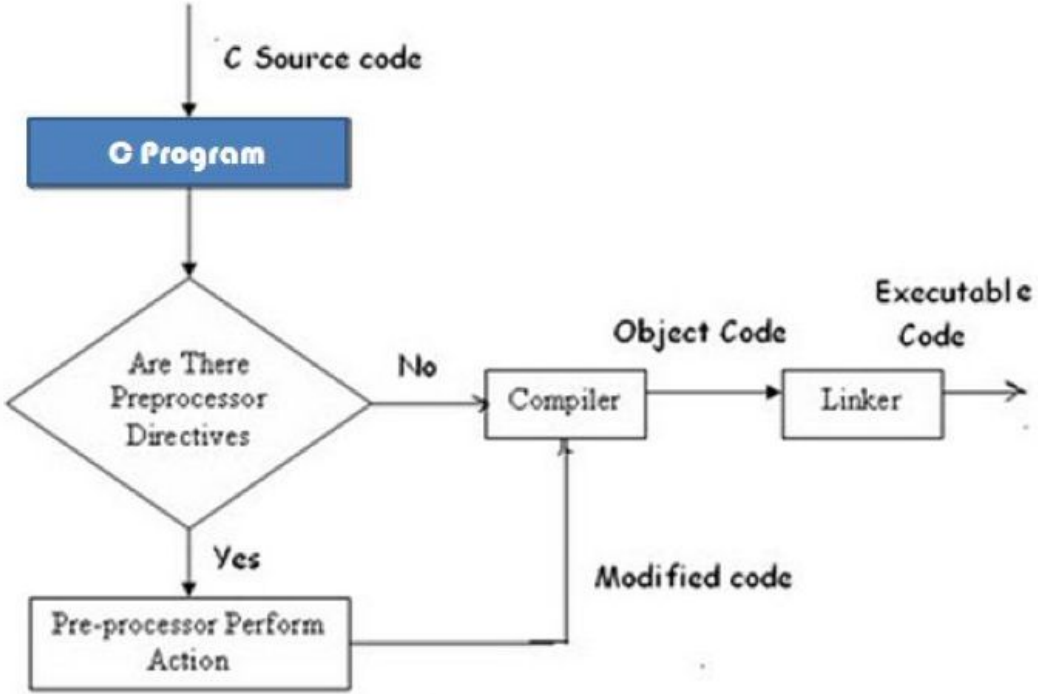
What is the PreProcessor?

- Suppose you have a local constant (local to the file) that you don’t want to make global, but you are using it throughout the file, so you don’t want to pass it as a parameter. Solution?
  - Suck it up, and pass it as a parameter?
  - Make it global and name it very carefully?
- Use a Macro
  - Allows you to create your own language constructs
  - Make direct changes to your source **before** compilation
  - Uses a separate syntax from C

Text Substitution

- Essentially, the Preprocessor is just a text substitution tool
  - Find/Replace
- All macros use preprocessing to find and replace
  - Find ‘cat’, replace with ‘dog
- 2 types of macro
  - Expressions
  - Code Blocks

Preprocessor in Compilation



Preprocessing statement

- The preprocessing statement always starts with # followed by a keyword without a space.
  - #include
- You can add a preprocessing statement anywhere in your code, but it must go on its own line.

#include

- You already use this
- Directive:
  - Find this file
  - Replace this directive with all the text in the file
- Two syntaxes
  - #include <library.h> //go to the systems library for the source
  - #include “MyFile.h” //look in the current folder for the file -- more on this later

#define

- Allows you to define names for constants
- Syntax
  - #define TRUE 1 //notice no equal sign
- Usage
  - if(TRUE == 1) …
  - after preprocessing, but before compilation this becomes if(1 == 1) …
- Still just text replacement

#define Style Guide

- Must be defined before referenced
  - int var = MY_DEFINE //wrong!
    #define MY_DEFINE 1
  - The preprocessor is still sequential
- Define after #includes
- Use all CAPS and underscore for spaces
- ANY constant or repeated literal local value should use a #define
  - Example: #define PI 3.14159265359

Macros

- You can also #define expressions
  - #define THIRD 1/3
    int slice = 6 * THIRD
  - #define RIGHT_SHIFT_BYTE >> 8
    int val = 256 RIGHT_SHIFT_BYTE //what is val?
  - #define EQUALS ==
    if(x EQUALS y) …
    - *helps with debugging*

Macro Continuation

- Macros must be contained on one line
  - Text replacement cannot cross line breaks
- To continue to another line, use “\”
- Example:
  - ```
    #define  GREETING  \
            printf("Hello!\n")
    ```
    ```

Macro Arguments

- You can send arguments to macros just like a function
- Syntax:
  - `#define HALF(x) x/2`
- Usage:
  - `int pie_portion = HALF(6);`

Macro Gotchas

- Still just text replacement
  - No logic to the preprocessor, just find and replace
- Order of operations (again)
  - #define SQUARE(x) x*x;
    int val = SQUARE(1+1)
  - What happens when we do this?

Solution, as always, Parenthesis

- Always isolate your parameters and expression with parenthesis
- #define SQUARE(x) (x)*(x)
  - SQUARE(1+1) becomes
  - (1+1) * (1+1) = 4

Classwork: PreProcessing

Problems with Macro scope

- Given the following macro:
  - doubleinc(a,b) \
    (a)++; \
    (b)++;
- What happens if we use the macro thusly:
  - if(x > y)
    doubleinc(x, y);
    - *Expands to:*
      - if(x > y)
        (x)++;
        (y)++;

Isolating your Macros

- Always wrap your macros in curly braces
  - Eliminates accidental control flow problems
  - Allows intermediate variables without naming conflicts
- Scoped Macro
  - ```
    doubleinc(a,b) {      \
        (a)++;            \
        (b)++;            \
    }
    ```
    ```

Making More Robust Macros

- You can use # in a macro to stringify arguments
  - This allows you to create some nice debugging macros
    - *#define peval(cmd, f) printf(#cmd ": " f "\n", cmd);*
- Preprocessor Strings
  - The preprocessor automatically concatenates adjacent strings
    - *That is why we can write: #cmd ": " f "\n"*
      - Each part is a string, and gets turned into one string

Predefined Macros

- Predefined macros
  - __DATE__
  - __TIME__
  - __FILE__
  - __LINE__

#ifdef

- #ifdef checks if a term has been defined
  - If true, the code is included in compilation
  - If false, code is not included in compilation
- Does not check the value of the definition
  - Value can be defined as 0 and still be defined, i.e. not boolean
    - *#define FOO 0*
      #ifdef FOO
      int x = 3;
      #endif
- This only gets included in compilation if FOO is defined

#endif

- All conditional preprocessor statements must end in #endif
  - Like the closing brace on a code block
- #ifdef FOO

  ….

  ….

  #endif

#ifndef

- Preprocessor can also check the inverse
  - If a name is not defined, execute code
  - Example
    - ```
      #define FOO 0
      #ifndef FOO
      int x = 3; //won't get executed
      #endif
      ```
      ```

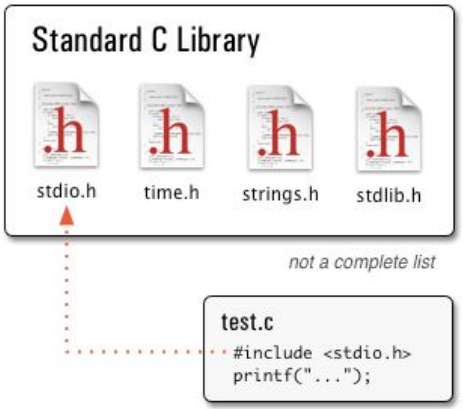
#if, #elif, #else

Additional Macro Statements

- #if -- Checks the value, i.e. is boolean
  - Example
    - ```
      #define FOO 0
      #if FOO
      int x = 3; //won't get executed
      #endif
      ```
- #else -- Preprocessor equivalent of 'else'
- #elif -- Preprocessor equivalent of 'else if'
      ```

Header Files

- Global Declarations (functions, type declarations, etc) should always go in a header file
  - Header files use the .h extension, <filename>.h
  - include header files with the #include preprocessor directive
- Library and User Created header files
  - Use header files to ‘modularize’ your code
  - Do not put executable code in a header file



File Inclusion

MyData.h

```
int x=5;
int y=5;
```

MyProg.c

```
#include <stdlib.h>
#include “MyData.h”
int main(){
   int z = x + y;
}
```

File Inclusion Result

```
#include <stdlib.h>

int x=5;
int y=5;

int main(){
   int z = x + y;
}
```

What if?

xdata.h

```
#include “ydata.h”
 int x = 5;
```

ydata.h

```
#include “xdata.h”
int y=5;
```

MyProg.c

```
#include <stdlib.h>
#include “xdata.h”
int main(){
   int z = x + y;
}
```

Infinite Loop Include

```
#include <stdlib.h>
int x = 5;
int y = 5;
int x = 5;
int y = 5;
…
…
```

Include Guards

- Use include guards in all header files
  - ```
    #ifndef FILENAME_H
    #define FILENAME_H
    …
    //code
    …
    #endif
    ```
- Never, ever, include a source file (file.c)
- Only include what is absolutely necessary in header files
    ```

Classwork

Conditional Compilation

```
#define MAN(x, y, z) x > y ? 1 : z
```