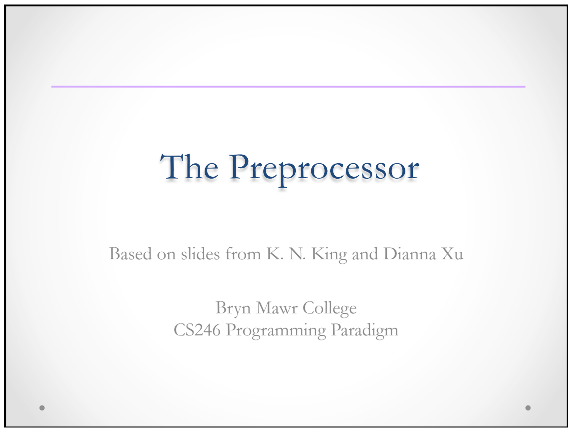
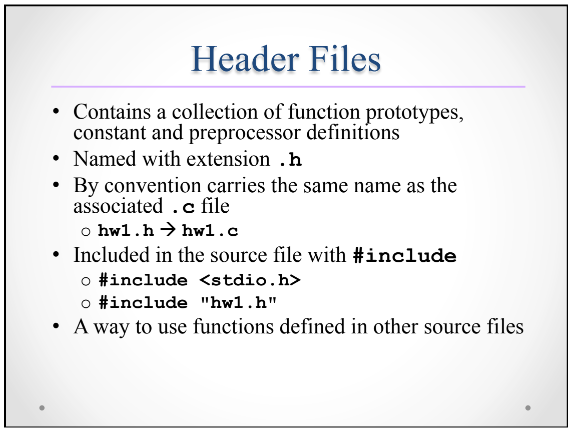


The Preprocessor

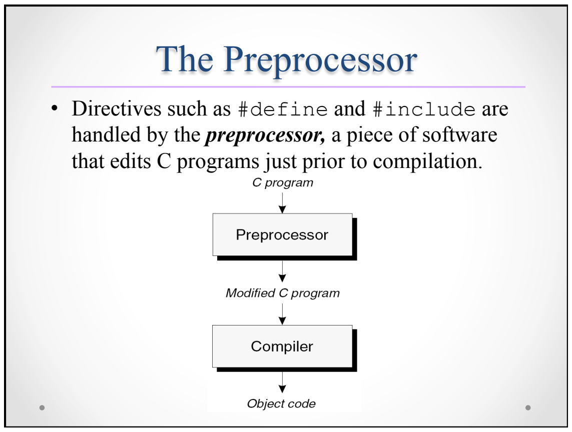
Based on slides from K. N. King and Dianna Xu

Bryn Mawr College
CS246 Programming Paradigm



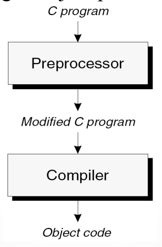
Header Files

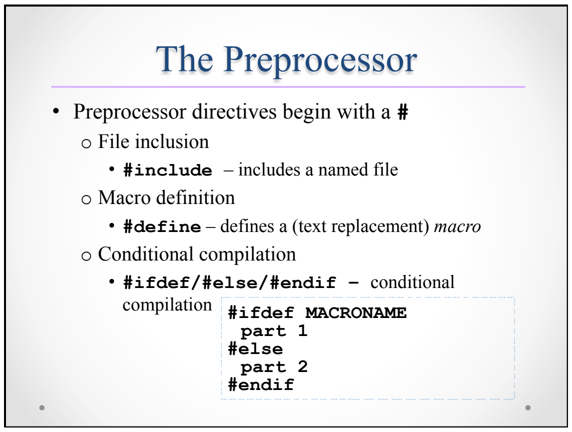
- Contains a collection of function prototypes, constant and preprocessor definitions
- Named with extension **`.h`**
- By convention carries the same name as the associated **`.c`** file
  - **`hw1.h`** → **`hw1.c`**
- Included in the source file with **`#include`**
  - **`#include <stdio.h>`**
  - **`#include "hw1.h"`**
- A way to use functions defined in other source files



The Preprocessor

- Directives such as `#define` and `#include` are handled by the ***preprocessor,*** a piece of software that edits C programs just prior to compilation.

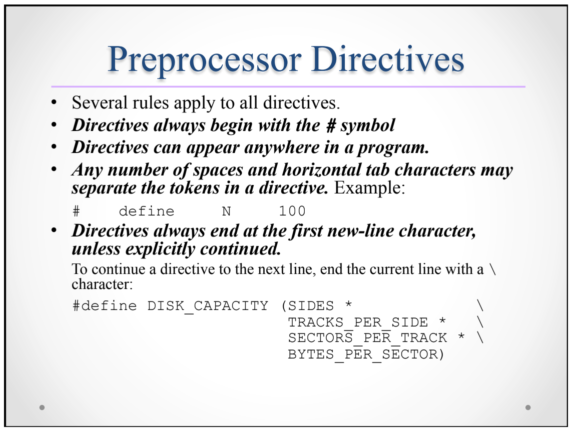




The Preprocessor

- Preprocessor directives begin with a **#**
  - File inclusion
    - **`#include`** – includes a named file
  - Macro definition
    - **`#define`** – defines a (text replacement) *macro*
  - Conditional compilation
    - **`#ifdef/#else/#endif`** – conditional compilation

```
#ifdef MACRONAME
  part 1
#else
  part 2
#endif
```



Preprocessor Directives

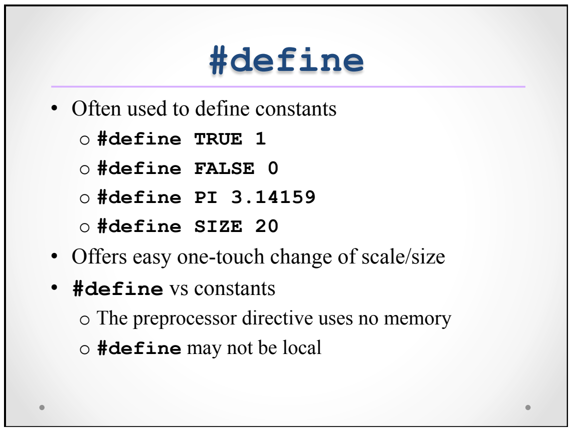
- Several rules apply to all directives.
- ***Directives always begin with the # symbol***
- ***Directives can appear anywhere in a program.***
- ***Any number of spaces and horizontal tab characters may separate the tokens in a directive.*** Example:

```
#    define     N     100
```

- ***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:

```
#define DISK_CAPACITY (SIDES *             \
                       TRACKS_PER_SIDE *   \
                       SECTORS_PER_TRACK * \
                       BYTES_PER_SECTOR)
```



`#define`

- Often used to define constants
  - **`#define TRUE 1`**
  - **`#define FALSE 0`**
  - **`#define PI 3.14159`**
  - **`#define SIZE 20`**
- Offers easy one-touch change of scale/size
- **`#define`** vs constants
  - The preprocessor directive uses no memory
  - **`#define`** may not be local

#define - more readable

```
#include<stdio.h>
#define MILE 1
#define KM   2

void km_mile_conv(int choice) {
  // …
  if (choice == MILE)
  // …
}
int main() {
  // …
  switch (choice) {
  case MILE:
    km_mile_conv(choice);
    break;
  caea KM:
    km_mile_conv(choice);
    break;
  /* more cases */
  }
}
```

Parameterized Macros

- Examples of parameterized macros:
```
#define MAX(x,y)    ((x)>(y)?(x):(y))
#define IS_EVEN(n)  ((n)%2==0)
```
- Invocations of these macros:
```
i = MAX(j+k, m-n);
if (IS_EVEN(i)) i++;
```
- The same lines after macro replacement:
```
i = ((j+k)>(m-n)?(j+k):(m-n));
if (((i)%2==0)) i++;
```
- A more complicated function-like macro:
```
#define TOUPPER(c) \
  ('a'<=(c)&&(c)<='z'?(c)-'a'+'A':(c))
```

Parameterized Macros

- ***A macro may evaluate its arguments more than once.***
  Unexpected behavior may occur if an argument has side effects:
```
n = MAX(i++, j);
```
  The same line after preprocessing:
```
n = ((i++)>(j)?(i++):(j));
```
- Errors caused by evaluating a macro argument more than once can be difficult to find, because a macro invocation looks the same as a function call.
- For self-protection, it's a good idea to avoid side effects in arguments.

The # Operator

- *The # operator converts a macro argument into a string literal*; it can appear only in the replacement list of a parameterized macro.
- For example:
```
#define PRINT_INT(n) printf(#n " = %d\n", n)
```
- The invocation
```
PRINT_INT(i/j);
```
  will become
```
printf("i/j" " = %d\n", i/j);
```
- The compiler automatically joins adjacent string literals, so this statement is equivalent to
```
printf("i/j = %d\n", i/j);
```

The ## Operator

- The ## operator can "paste" two tokens together to form a single token.
- If one of the operands is a macro parameter, pasting occurs after the parameter has been replaced by the corresponding argument.
- A macro that uses the ## operator:
```
#define MK_ID(n) i##n
```
- A declaration that invokes `MK_ID` three times:
```
int MK_ID(1), MK_ID(2), MK_ID(3);
```
- The declaration after preprocessing:
```
int i1, i2, i3;
```

General Properties of Macros

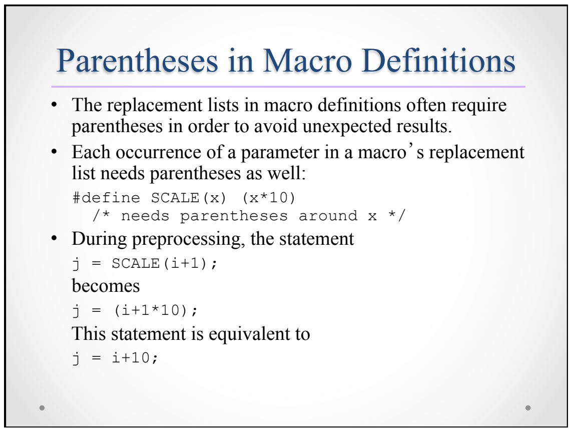
- ***Macros may be "undefined" by the `#undef` directive.***

  The `#undef` directive has the form

  `#undef` *identifier*

  where *identifier* is a macro name.

  One use of `#undef` is to remove the existing definition of a macro so that it can be given a new definition.



Parentheses in Macro Definitions

- The replacement lists in macro definitions often require parentheses in order to avoid unexpected results.
- Each occurrence of a parameter in a macro's replacement list needs parentheses as well:

```
#define SCALE(x)  (x*10)
  /* needs parentheses around x */
```

- During preprocessing, the statement

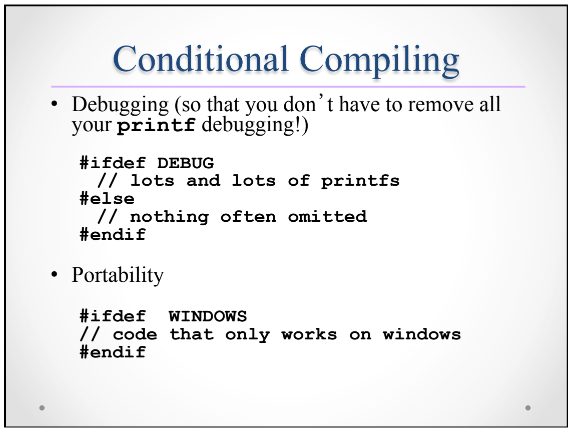
```
j = SCALE(i+1);
```

becomes

```
j = (i+1*10);
```

This statement is equivalent to

```
j = i+10;
```



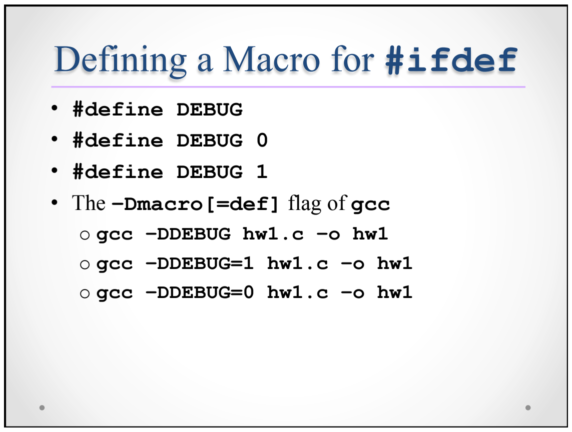
Conditional Compiling

- Debugging (so that you don't have to remove all your **printf** debugging!)

```
#ifdef DEBUG
  // lots and lots of printfs
#else
  // nothing often omitted
#endif
```

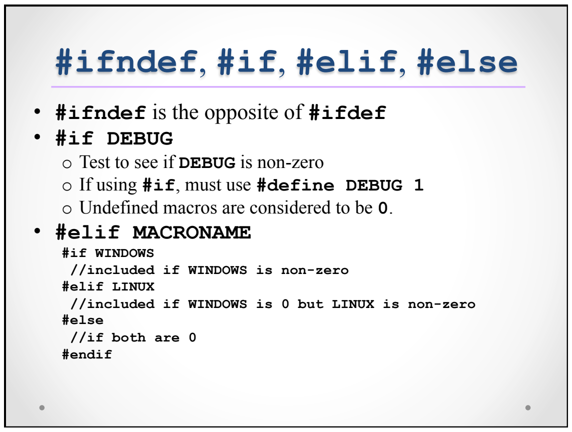
- Portability

```
#ifdef  WINDOWS
// code that only works on windows
#endif
```



Defining a Macro for #ifdef

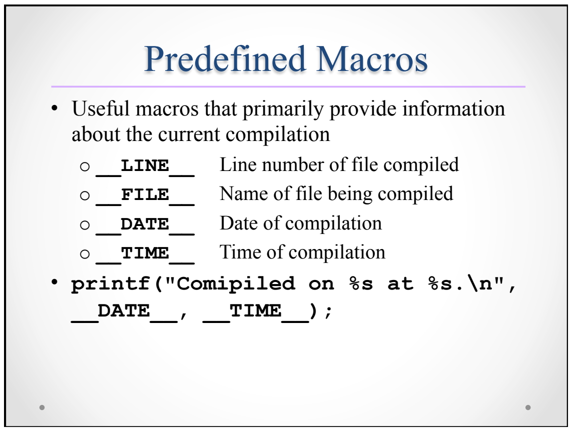
- **#define DEBUG**
- **#define DEBUG 0**
- **#define DEBUG 1**
- The **-Dmacro[=def]** flag of **gcc**
  - **gcc -DDEBUG hw1.c -o hw1**
  - **gcc -DDEBUG=1 hw1.c -o hw1**
  - **gcc -DDEBUG=0 hw1.c -o hw1**



#ifndef, #if, #elif, #else

- **#ifndef** is the opposite of **#ifdef**
- **#if DEBUG**
  - Test to see if **DEBUG** is non-zero
  - If using **#if**, must use **#define DEBUG 1**
  - Undefined macros are considered to be **0**.
- **#elif MACRONAME**

```
#if WINDOWS
 //included if WINDOWS is non-zero
#elif LINUX
 //included if WINDOWS is 0 but LINUX is non-zero
#else
 //if both are 0
#endif
```

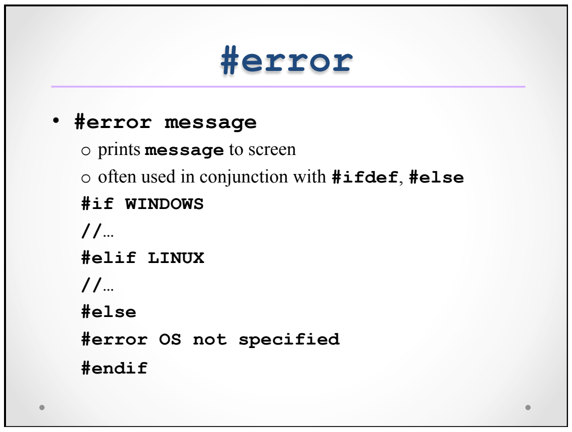


Predefined Macros

- Useful macros that primarily provide information about the current compilation

__LINE__	Line number of file compiled
__FILE__	Name of file being compiled
__DATE__	Date of compilation
__TIME__	Time of compilation

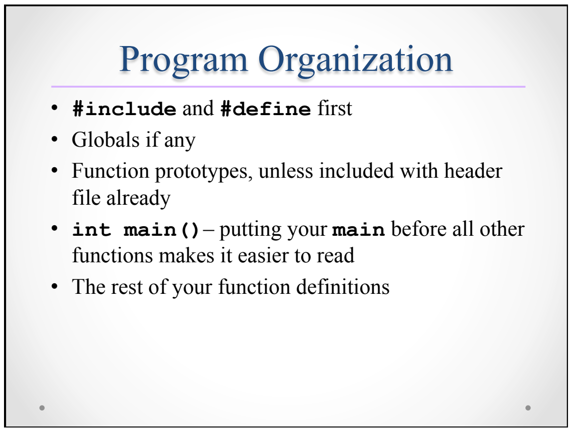
-
  ```
  printf("Comipiled on %s at %s.\n",
  __DATE__, __TIME__);
  ```



#error

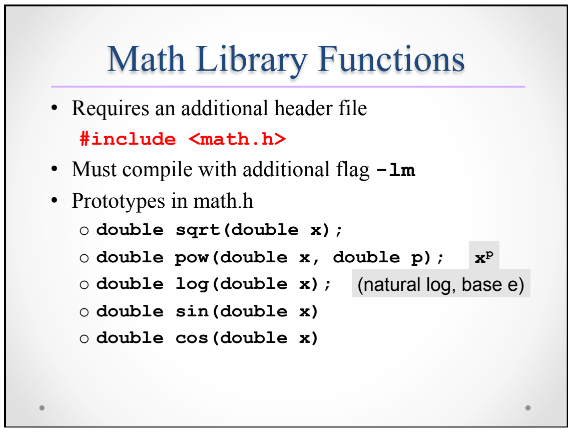
- **#error message**
  - prints **message** to screen
  - often used in conjunction with **#ifdef**, **#else**

```
#if WINDOWS
//…
#elif LINUX
//…
#else
#error OS not specified
#endif
```



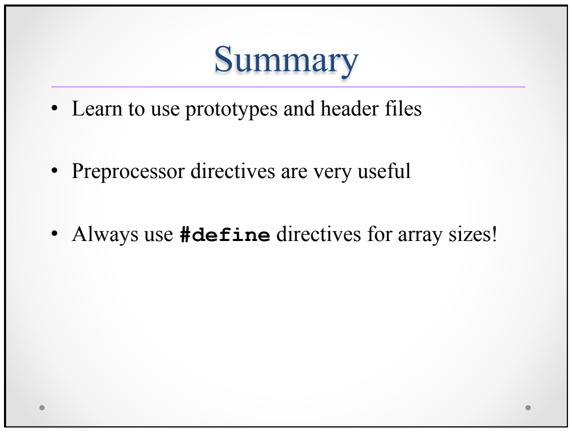
Program Organization

- **`#include`** and **`#define`** first
- Globals if any
- Function prototypes, unless included with header file already
- **`int main()`** – putting your **`main`** before all other functions makes it easier to read
- The rest of your function definitions



Math Library Functions

- Requires an additional header file
  `#include <math.h>`
- Must compile with additional flag **`-lm`**
- Prototypes in math.h
  - **`double sqrt(double x);`**
  - **`double pow(double x, double p);`** x^p
  - **`double log(double x);`** (natural log, base e)
  - **`double sin(double x)`**
  - **`double cos(double x)`**



Summary

- Learn to use prototypes and header files
- Preprocessor directives are very useful
- Always use **`#define`** directives for array sizes!