

# CSC180: Lecture 24

Wael Aboulsaadat

wael@cs.toronto.edu

<http://portal.utoronto.ca/>

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

# C-String functions: sscanf and sprintf

- Syntax :

  - sscanf (instring, “format”, variable\_address);

  - Similar to scanf, it inputs data from string instead of keyboard
  - Format: similar to those used in printf, see Chapter 4
  - If use %s format, it reads all until it reaches the first white space

- Syntax:

  - sprintf (outstring, “format”, variable);

  - Similar to printf, it outputs results to string instead of screen

# C-String functions: sscanf and sprintf

```
char    c;  
int     i;  
double db;  
char    s[80], outstring[80],  
        *instring="a 100 -1.23 This is a sample data";  
  
sscanf(instring, "%c%d%lf%s", &c, &i, &db, s);  
//Then c='a', i=100, db=-1.23, s="This"  
  
sprintf(outstring, "%c%d%lf%s", c, i, db, s);  
// print to outstring: a 100 -1.23 This
```

# C-String functions: strtok

- Syntax :

`strtok(instring, delimiter)`

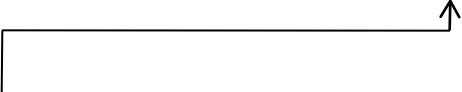
- Used to retrieve the tokens of a sentence...
- keeps an internal pointer to the string

# C-String functions: strtok

```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";  
  
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```

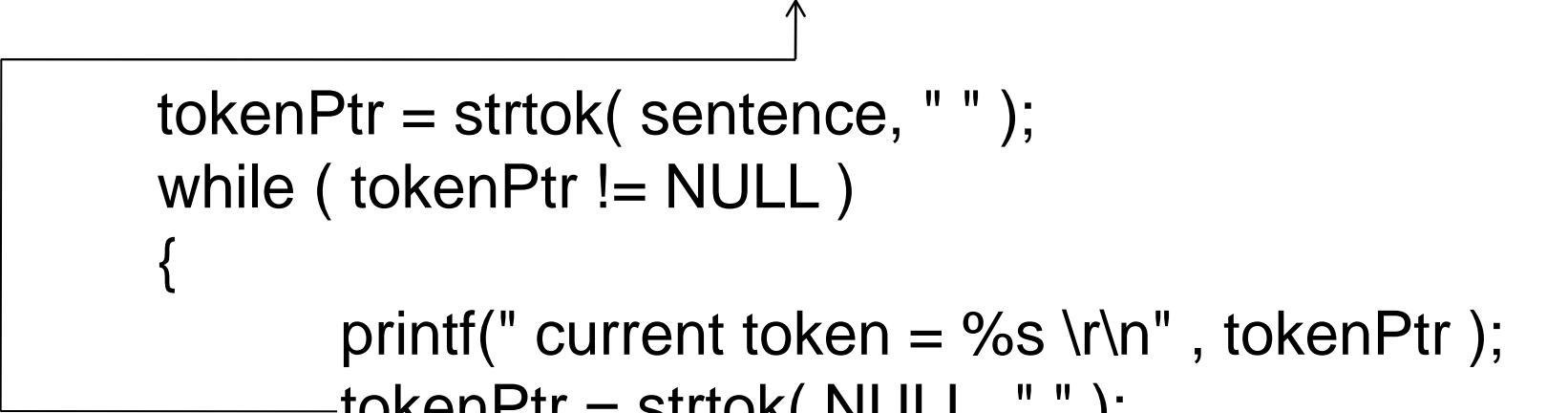
# C-String functions: strtok

```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";  
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```



# C-String functions: strtok

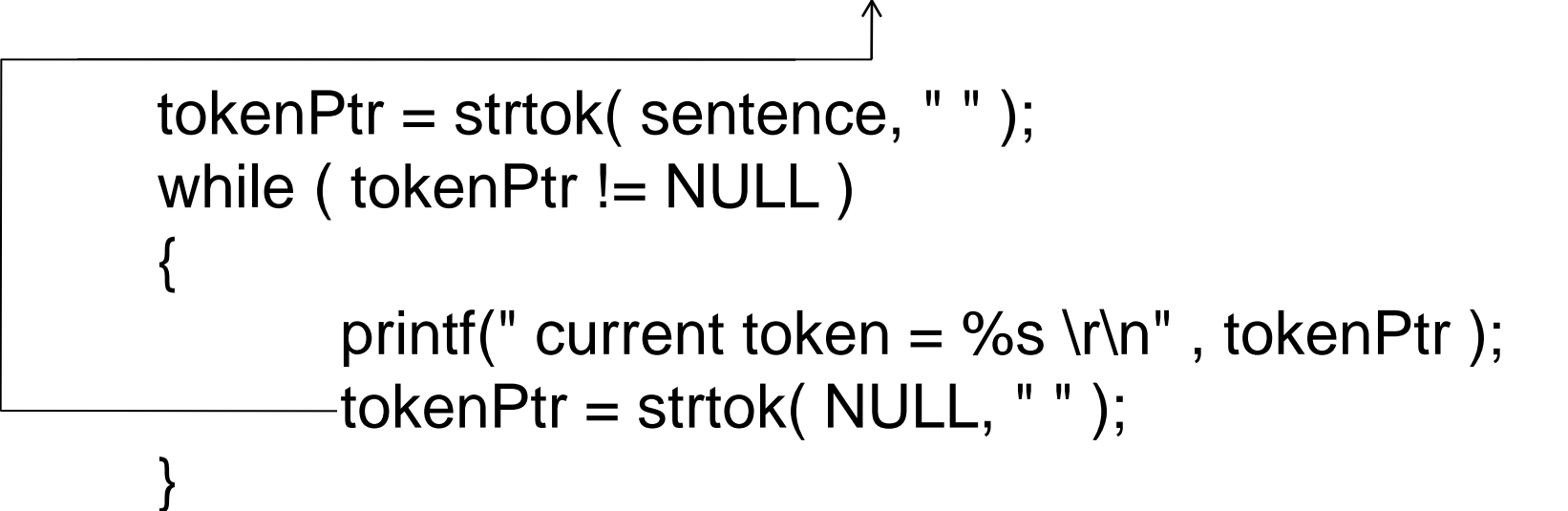
```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";
```



```
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```

# C-String functions: strtok

```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";
```

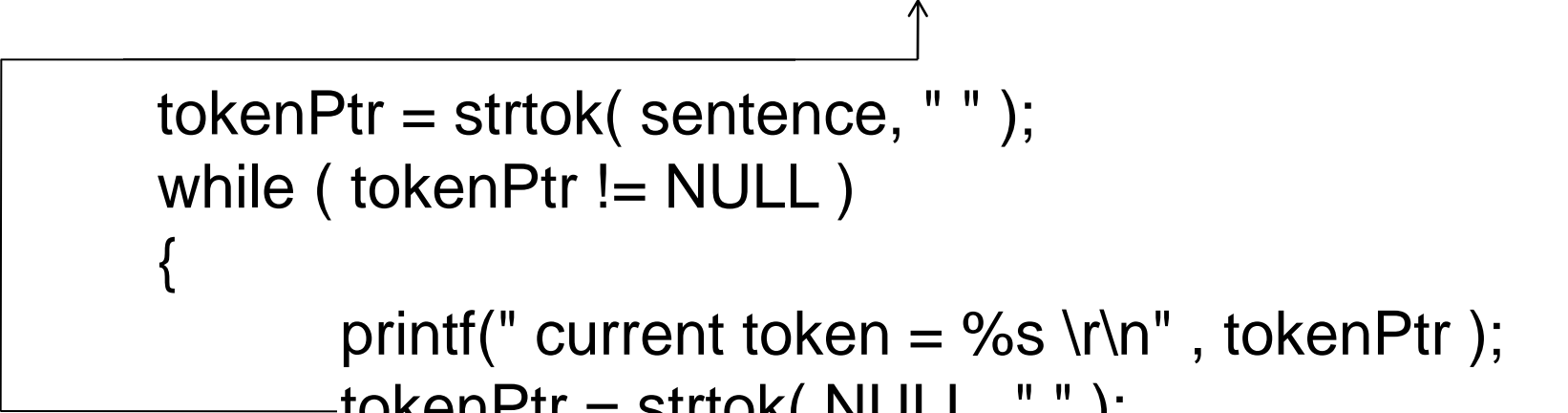


```
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```



# C-String functions: strtok

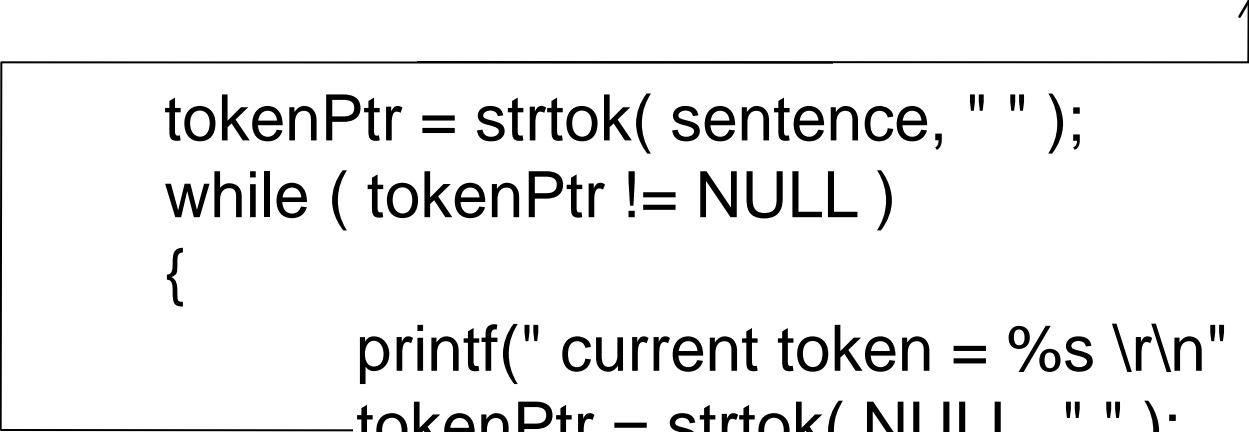
```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";
```



```
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```

# C-String functions: strtok

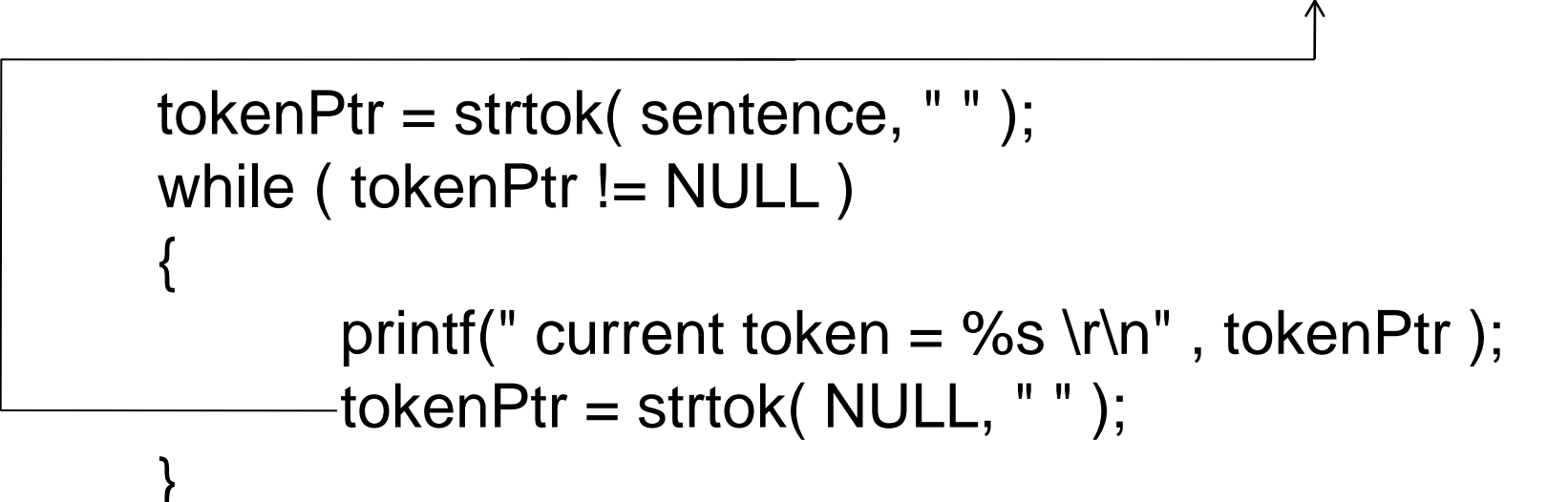
```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";
```



```
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```

# C-String functions: strtok

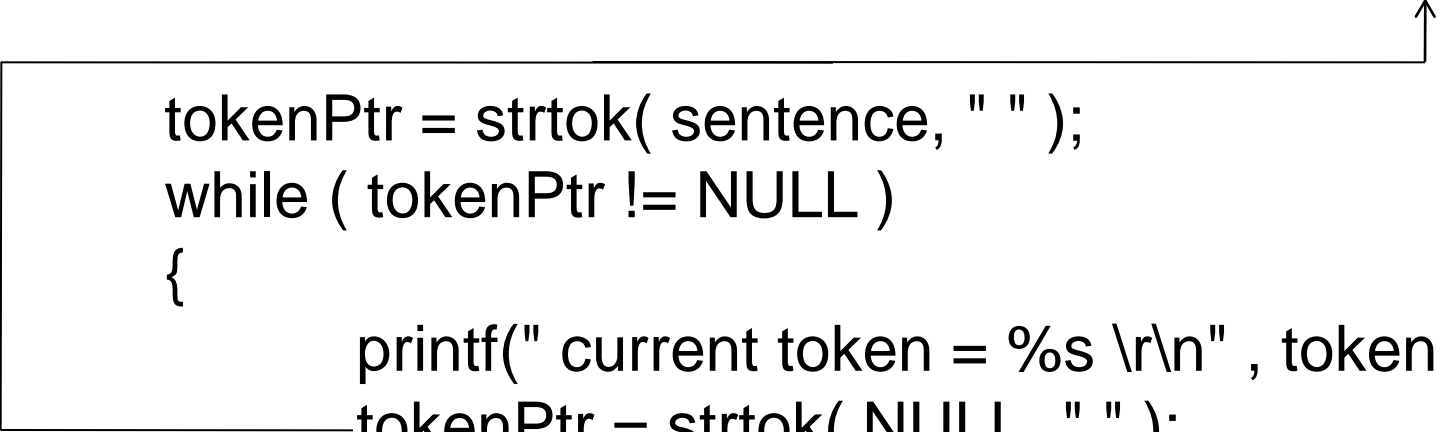
```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";
```



```
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```

# C-String functions: strtok

```
char *tokenPtr;  
char sentence[] = "This is a sentence with 7 tokens";
```

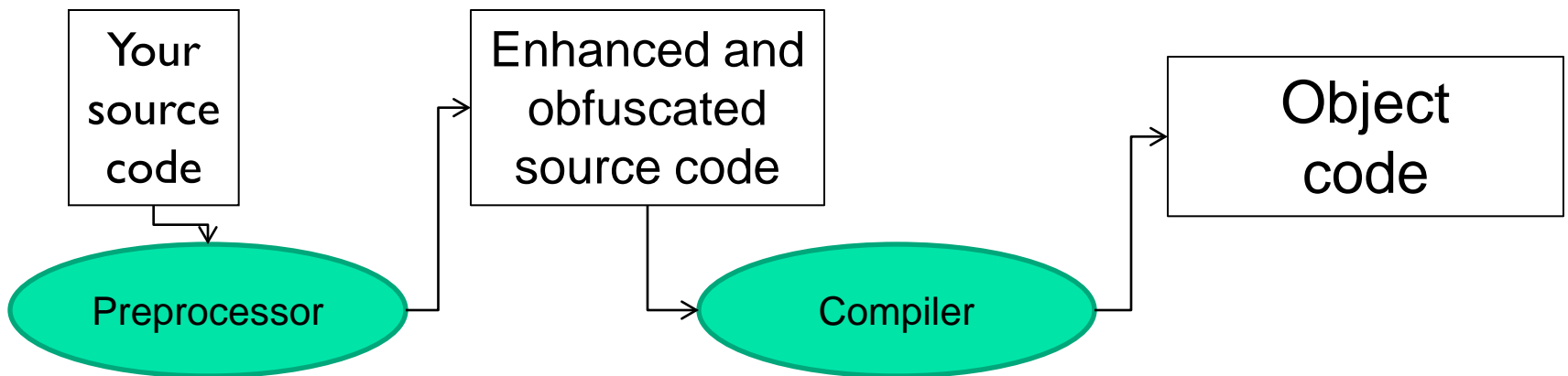


```
tokenPtr = strtok( sentence, " " );  
while ( tokenPtr != NULL )  
{  
    printf(" current token = %s \r\n" , tokenPtr );  
    tokenPtr = strtok( NULL, " " );  
}
```

Preprocessor

# The preprocessor

- ▶ The **preprocessor** takes your source code and – following certain **directives** that you give it – tweaks it in various ways before compilation.
- ▶ A directive is given as a line of source code starting with the # symbol
- ▶ The preprocessor works in a very crude, “word-processor” way, simply cutting and pasting – it doesn’t really know anything about C!



# Preprocessor Directives: rules

- The Must begin with a #
- May contain extra spaces and tabs
- End at the first new-line character, unless continued using \

# Preprocessor directives: #define value

- The `#define` directives perform “global replacements”:

```
#define MAX_COLS 20  
#define MAX_INPUT 1000
```

- In your code, every instance of `MAX_COLS` is replaced with `20`, and every instance of `MAX_INPUT` is replaced with `1000`.



# Preprocessor directives: #define value

```
#define N 100
```

```
#define PI 3.14159
```

```
#define WARNING_MSG "Warning: nonstandard feature"
```

```
#define BEGIN {
```

```
#define END }
```

```
#define BOOL int
```

```
if ( nIndex < N )  
BEGIN  
    printf( "%s", WARNING_MSG );  
END
```

This is what you see

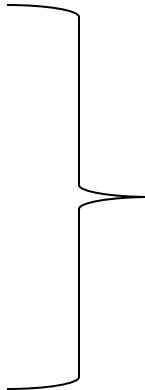
```
if ( nIndex < 100 )  
{  
    printf( "%s", "Warning: nonstandard  
        feature" );  
}
```

This is what the compiler see/compile

# Preprocessor directives: #define macro

```
#define CUBE(x) ((x)*(x)*(x))
```

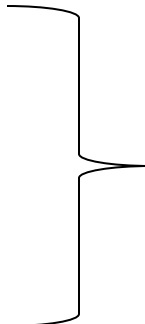
```
if (i==3)  
    x=CUBE(y+2);  
else  
    x=y+2;
```



This is your code/  
what you see

The above code is expanded to:

```
If (i==3)  
    x=((y+2)*(y+2)*(y+2));  
else  
    x=y+2;
```



This is what the  
compiler see/compile

# Preprocessor directives: #define pitfalls

- Bad ☹️

```
#define mymult(a, b) a * b
```

```
...
```

```
k = mymult( (i-1), (j+5);
```

```
k = i - 1 * j + 5;
```

This is your code/  
what you see

This is what the  
compiler see/compile

- Better 😊

```
#define mymult(a, b) (a) * (b)
```

```
.....
```

```
k = mymult(i-1, j+5);
```

```
k = (i - 1) * (j + 5);
```

This is your code/  
what you see

This is what the  
compiler see/compile

# Preprocessor directives: #define pitfalls

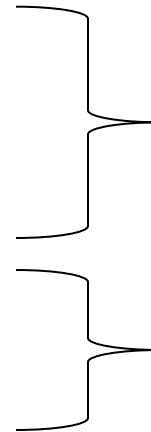
- Be careful of other side effects, for example what if we did the following ? ☹️

```
#define mysq(a) a * a
```

```
....
```

```
k = mysq(i++)
```

```
k = i++ * i++
```



This is your code/  
what you see

This is what the  
compiler see/compile

# Preprocessor directives: #include

/usr/include/stdio.h

```
/* comments */
#ifndef _STDIO_H
#define _STDIO_H

... definitions and protoypes

#endif
```

/usr/include/stdlib.h

```
/* comments */
#ifndef _STDLIB_H
#define _STDLIB_H

... definitions and
protoypes

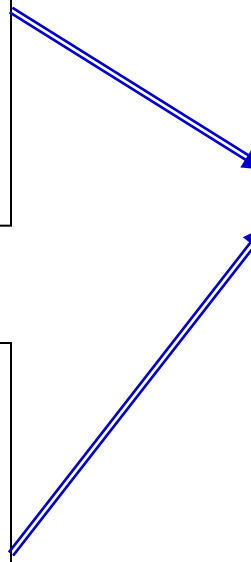
#endif
```

#include directs the preprocessor to “include” the contents of the file at this point in the source file.

example.c

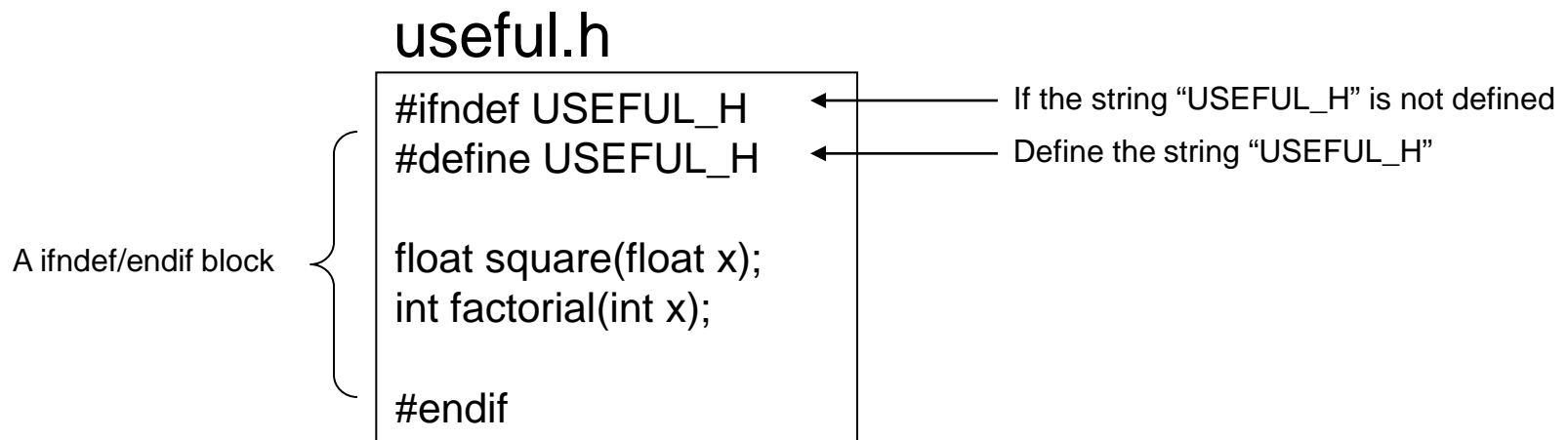
```
#include <stdio.h>
#include <stdlib.h>
/* other includes */

int main()
{
    printf(“Hello world” );
    return 0;
}
```



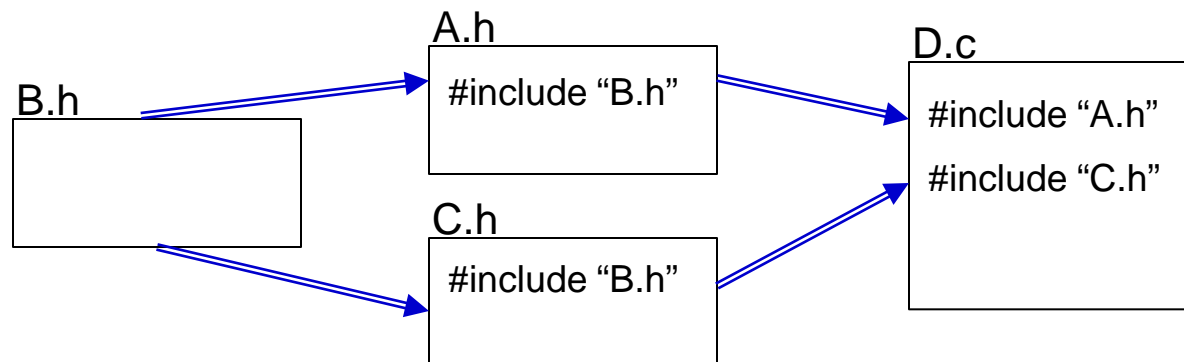
# #ifndef, #define and #endif

- “#define” defines a symbol.
- #define ABC defines the string “ABC”.
  - Now, the string “ABC” is recognized by the pre-processor.
- “#ifndef ABC” means “if the string ABC is not defined”
- “#endif” closes the “#ifndef” block.



# #ifndef in header files (\*.h)

- The `#ifndef`, `#define` and `#endif` lines are used in header files to prevent them from being included multiple times.
- E.g.
  - A.h includes B.h, and C.h also includes B.h.
  - Then, D.c includes both A.h and C.h.
    - In this case, B.h is included twice in D.c.



# #ifndef in header files (\*.h)

- Without the `#ifndef` lines, the compiler would complain that functions are declared multiple times.
- With the `#ifndef` lines, the preprocessor would completely ignore B.h the second time it is included.

