ex. # undef SQUARE /* undefine SQUARE */

continued from Lecture 24

3) conditional inclusion

eg. #if 0
...
#endif
...
#else if x == 1
#elif x == 2

Calling this function:

gcc -ansi -W -Wall -pedantic -DOS=DOS clear.c

no spaces

We can also do:

#if defined (OS)
... /* if OS is defined */
#endif

#if !defined (OS) /* if OS is not defined */ #ifndef
...
#end if

eg. #ifdef DEBUG
fprintf (stderr, " ");
#endif

eg. Macros example

```
#define OTHER 3
#define DOS 1
#define UNIX 2

int main (void) {
  #if OS == DOS
    ...
  #elif OS == UNIX
    ...
  #else OS == OTHER
    ...
  #end if
}
```

gcc -DOS

eg.
```
#ifndef LINESIZE
   #define LINESIZE 1024 /* give LINESIZE = default value of 1024 */
#endif
```

Can override it using
```
gcc -DLINESIZE=512
```

Include guards:
```
#include <stdio.h>
#include <stdio.h>
```
} is stdio.h included twice??

No, include guads prevent a header file from being read in or included more than once.

eg.
```
/* save.h */
#ifndef SORT_H
#define SORT_H

   /* prototypes go here */
#endif
```

1st time: SORT_H is not defined yet, → it proceeds to define it.

Next time we #include "sort.h", SORT_H is already define and is skipped.

4) stringize

allows us the continuation of the statement to the next line

```
#define CHECK(PRED) printf("%s ... %s\n", (PRED)?"passed":"fail", \
   #PRED)
```

↑ converts PRED to string

▵ static

has 2 meanings:
1) A static local variable: lives through whole program
2) A static global variable: has static linkage
   or external
   or function

▵ Static local variable

```
int f(void){
   static int n;
   return ++n;
}
```

f() ⇝ 1
f() ⇝ 2
f() ⇝ 3

→ static local variable are automatically initialized to ∅

```
int g(void) {
  static int n=1;
  return n++;
}
```

g() ⇝ 1
g() ⇝ 2

▵ Static Linkage

- a function or an external variable has static linkage if it cannot be linked from another file.

```
/*file1.c*/
#include <stdio.h>
int main(void){
 printf("%d\n", square(1));
 return 0;
}
```

```
/*file2.c*/
int square(int x) {
   return x * x;
}
```

continued...

...Lecture 25

compile statements cont...

```
gcc -ansi -W -Wall -pedantic -c file1.c
gcc -ansi -W -Wall -pedantic -c file2.c
```

Link them together via: (object files)

```
gcc file1.o file2.o
```