2510 : Lecture 24 || March 24/17

▷ Assignment 3
- extra words in the record input are ignored

a12345678 homer simpson 35 go to hell
↑ ignored.

◦ continued from Lecture 23

```
void (*original (int, void (*)(int)))(int);
```
param — param 2

- signal returns a function pointer
- the 2nd param is also a function pointer of the same type.

```
typedef void (*sighandler_t)(int);
sighandler_t signal (int, sighandler_t);
```

▷ prototype of qsort

```
void qsort (void *, size_t, size_t, int (*)(const void *, const
                                                    void *));
```

▷ The C Preproccessor

When we compile our C program using gcc, it goes through several stages.

C source —preprocessor→ output (preprocessed) —compiler→ ~~assembly~~ object files
~~output~~
↓ linker
machine code

... Lecture 24

gcc -E file.c : stop after preprocessing
gcc -c file.c : stop after compiling doesn't link (object files output)

- The preprocessor handles preprocessor directives.
Some preprocessor directives:

1) #include 2 versions
eg. #include <stdio.h> /* for sys header files */
#include "a3.h" /* for own header files */

↳ difference in where the preprocessors look for the header files

eg. #include "headers/record.h"
↳ use forward slash

↳ essentially, the #include directives asks the preprocessor to read in the content of the header file

Note: Don't #include c source files!
Only #include header files

2) defining macros : #define
#define LINESIZE 1024
↳ macro or symbolic names

char line[LINESIZE]; ⟶ (macro expansion by preprocessor) char line[1024];

NOT COMPILER.

... Lecture 24

- macro expansion is basically done via text replacement

```
#define UPSIZE 1024;
char buffer [UPSIZE];   ⇝   char buffer [1024;];
```

↳ the preprocessor is smart enough not to replace macros when it is inside a string.
eg. "the LINESIZE is"

▷ function-like macros

✗
```
#define SQUARE (x)    x * x
int n = SQUARE(2);    ⇝  int n = 2*2;
int m = SQUARE (2+3); ⇝  int m = 2+3*2+3;
```

✗
```
#define SQUARE2(x)   (x)*(x)
int m = SQUARE2(2+3);  ⇝  int m = (2+3)*(2+3);
int x = 25/SQUARE2(5); ⇝  int x = 25/ 5*5 ;
```

✓
```
#define SQUARE3(x); ((x)*(x))
int x = 25/SQUARE3(5);  ⇝  int x = 25/((5)*(5));
```

Lesson: use brackets

```
int n = 3;
int m = SQUARE3 (n++); ⇝ int m = ((n++)*(n++));
```

(value is unspecified)

SQUARE3 evaluates its arguments more than once.