C-Week-11-Notes

L1 Intro to Macros in C

- text expansion
  define sometext someothertext
  "This file has sometext."
  ↳ "This file has someothertext."
  replace
- it helps in direct replacement of text.
- Macro expansion - compile time code manipulation.
- direct 1-1 replacement of text
  cpp - preprocessor - expand macros, recursively if needed
- text expansion if parameters
  #define ABS(x) x<0 ? -x : x
- parantheses - required for avoiding precedence

  #define ABS(x) ((x)<0 ? -(x) : (x))
- positional arguements
  #define MAX(a, b)
  1st arg. 2nd arg.

- expanded at compile time → copy of the code created, can create complex code (code generation)
- not reusable / shared code → recursion
- - evaluated at compile time.

L2 Eqs. of Macros 1

```
#define PI 3.14159
#define HW "Hello World"
#define SQR(x) ((x)*(x))
#define MAX(x,y) x>y ? x : y
```

L3 Eqs. of Macros 2

```
#define PRINTALL(A)
{
    ____
}
```

this is impt. so that it means, its from the same line.

other func^{ns} —

__ func __ → func^{n} name

- __FILE__ → present file name
- __LINE__ ⇒ particular line no.
- __DATE__ → today's date
- __TIME__ → current time
- __STDC_VERSION__ → this C version

L4 Include File in C

- include - file level macro expansion, complete text included - generates new source code, possible to include entire files of code (this is not advisable), header files (type and parameter def^n)
- #include recur.cc
  printf("Hello World!")

L5 Multifile Projects in C

- refactoring - separate out common code into functions, or sep. out common code into libraries, code reusability (testing, debugging made easier)
- declare once, use later → func^n

prototypes

- prototype needed to create func^n placeholder (func^n signature)
- static variables - file scope, different use than func^n static
- extern variables - externally defined variables, redeclara^n is not allowed, refer to variables from other file.
- linker error → when more than 1 file does not get linked.

L6 Compila^n Process of C Programs

- compile process

source object executable

.c → .o →
.c → .o → exe
.c → .o →

compile

C source → machine code

↳ link
→ combine libraries
→ possibly diff. origin (and allocate relative addresses

- compilaⁿ commands
  gcc -c hello.c → hello.o
  gcc hello.o → a.out / hello
- collect commands into file → execute sequentially
- problem — large projects, 1000s of files (incremental compilaⁿ)
- make (makefile) → track file changes, identify dependencies recompile only as needed, parallel compilaⁿ, conditional task execuⁿ / sequencing
- clean:
  rm -f main main-debug