CS107 Lecture 12

Preprocessor, Compilation, Linking

```
# define  kWidth  40
# define  kHeight 80
# define  kPerimeter  2 * (kWidth + kHeight)
```

substitutes text

macro

```
# define  MAX(a, b)  (((a) > (b)) ? (a) : (b))
```

no space ← parentheses to eliminate evaluation ambiguity

During preprocessing

MAX(10, 40) is replaced with ((10) > (40)) ? (10) : (40) as text

⇒ avoid call and return overhead of assembly function call (i.e. allocating param space, local space, copying return address and deallocation...)

↳ inline text

reduces use of stack

and # of assembly instructions associated with a function

no type checking by preprocessor

e.g. `int max = MAX(fib(100), fact(4000));`

e.g.
```
# define NthElemAddr(base, elemSize, index)  \
        ((char *)base + index * elemSize)
void *VectorNth(vector *v, int position)
{
    assert(position >= 0);
    assert(position < v->logLength);
    return NthElemAddr(v->elems, v->elemSize, position);
}
```

line cont.

asserts are also macros

can be stripped at compile time

assert in assert.h

```
#define assert(cond) \
  (cond) ? ((void)0) :
    fprintf(stderr, "_____"), exit(0)
```

full assert def:

← if passed of defined, then second assert replaced by first.

```
#ifdef NDEBUG
  #define assert(cond) (void)0
#else
  #define assert(cond) \
    (cond) ? ((void)0) :
      fprintf(stderr, "_____"), exit(0)
#endif
```

→ no assert failing in production code

Drawback:

```
int max = MAX(fib(100), fact(4000))
```

expands to

```
int max = (fib(100) > fact(4000)) ?
            fib(100) : fact(4000);
```

bad runtime

```
int larger = MAX(m++, n++);
```

expands to

```
int larger = ((m++) > (n++)) ? (m++) : (n++)
```

false (unintended) return value

lecture 12 CS 107

#include <stdio.h>
#include <assert.h>
#include "re.h" ← current working dir

} search and replace

replaces an #include line with the contents of file

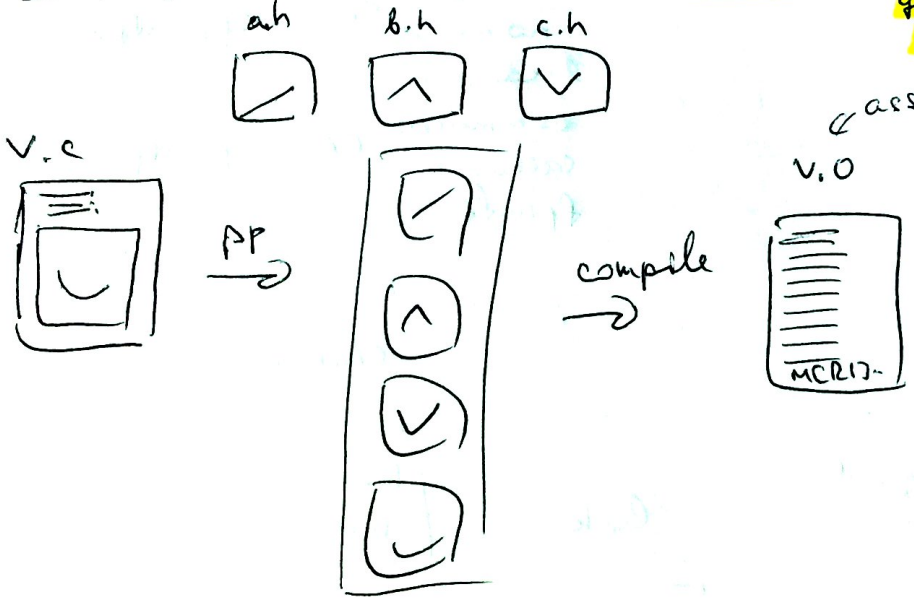
#include is recursive

gcc -E v.c
stop at preprocessor

```
#ifndef _VECTOR_H_
#define _VECTOR_H_
   [redacted]
#endif
```

make sure vector.h is not included twice

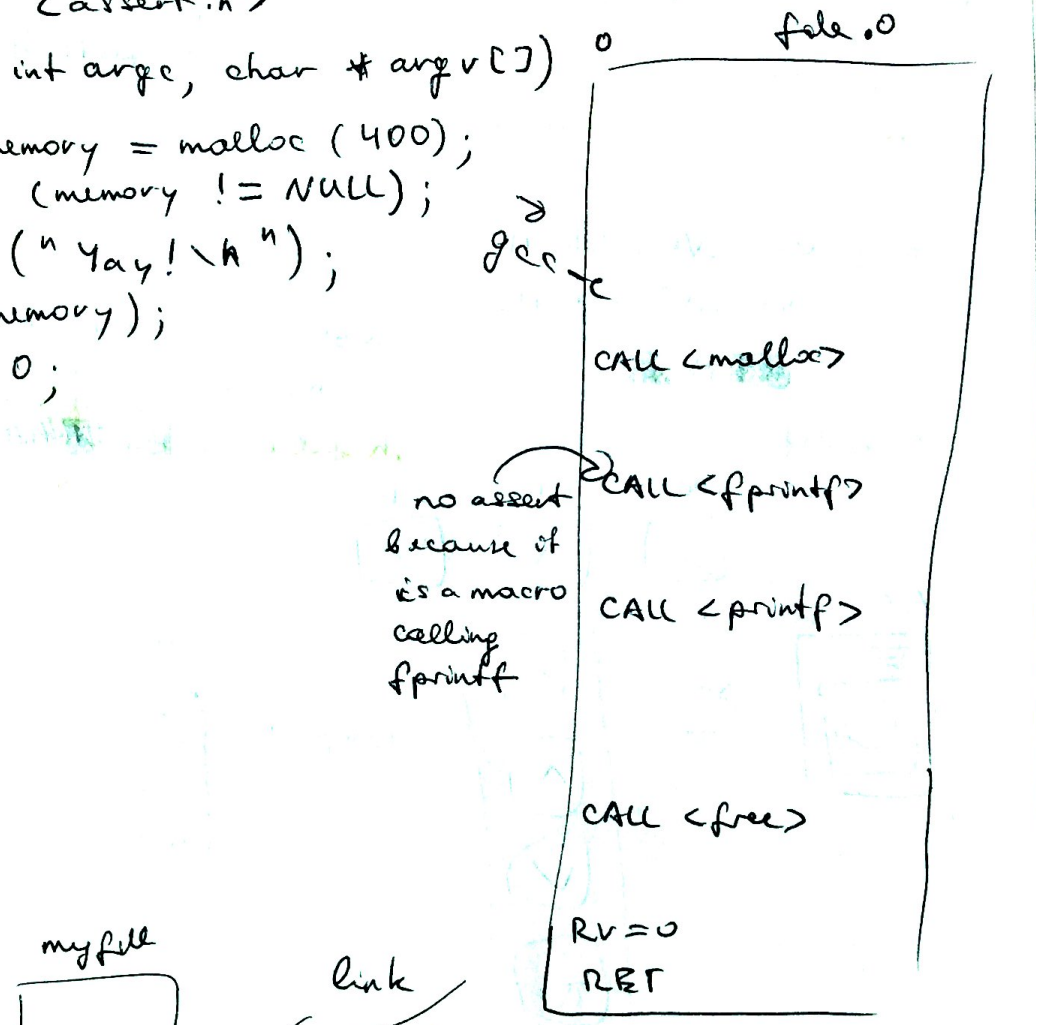
.h files only declaration [redacted] guarantee no code generation, no definitions

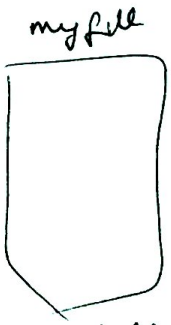


main.c

```
#include <stdio.h>  // printf
#include <stdlib.h> // malloc, free
#include <assert.h>
int main (int argc, char *argv[])
{
    void *memory = malloc (400);
    assert (memory != NULL);
    printf ("Yay!\n");
    free (memory);
    return 0;
}
```

gcc -c
stop at compilation





executable produced after linking

link
gcc -o myfile

malloc implementation ect..

Linking reqs: (linking .o files)

- need main function
- definition for every fxn that can be called
- each function can only be defined once