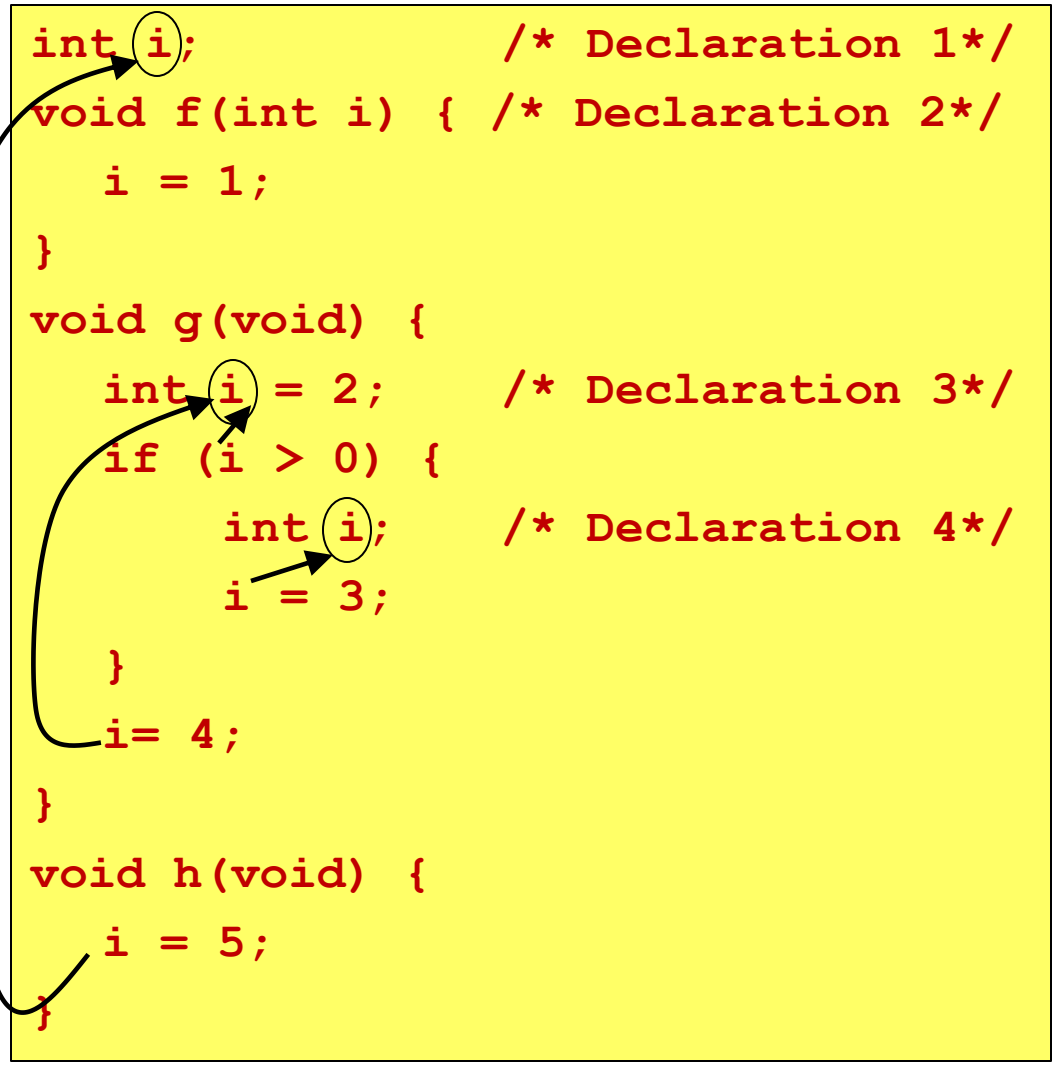
More on Scope, Pointers & Miscellaneous Topics

Goals of this Lecture

- Help you learn:
  - Extern Variables & Functions (File-level scope)
  - "Weird, Scary" Pointers
  - Miscellaneous Topics (casting, typedef *etc.*)

- Why? When creating large programs these techniques help maintain
  - Modularity across different object module files
  - Abstraction
  - Program comprehensibility and development ease

Revision - Scope



```
int i;              /* Declaration 1*/
void f(int i) {     /* Declaration 2*/
   i = 1;
}
void g(void) {
   int i = 2;       /* Declaration 3*/
   if (i > 0) {
        int i;      /* Declaration 4*/
        i = 3;
   }
   i= 4;
}
void h(void) {
   i = 5;
}
```

File-level scope
(Declaration 1)

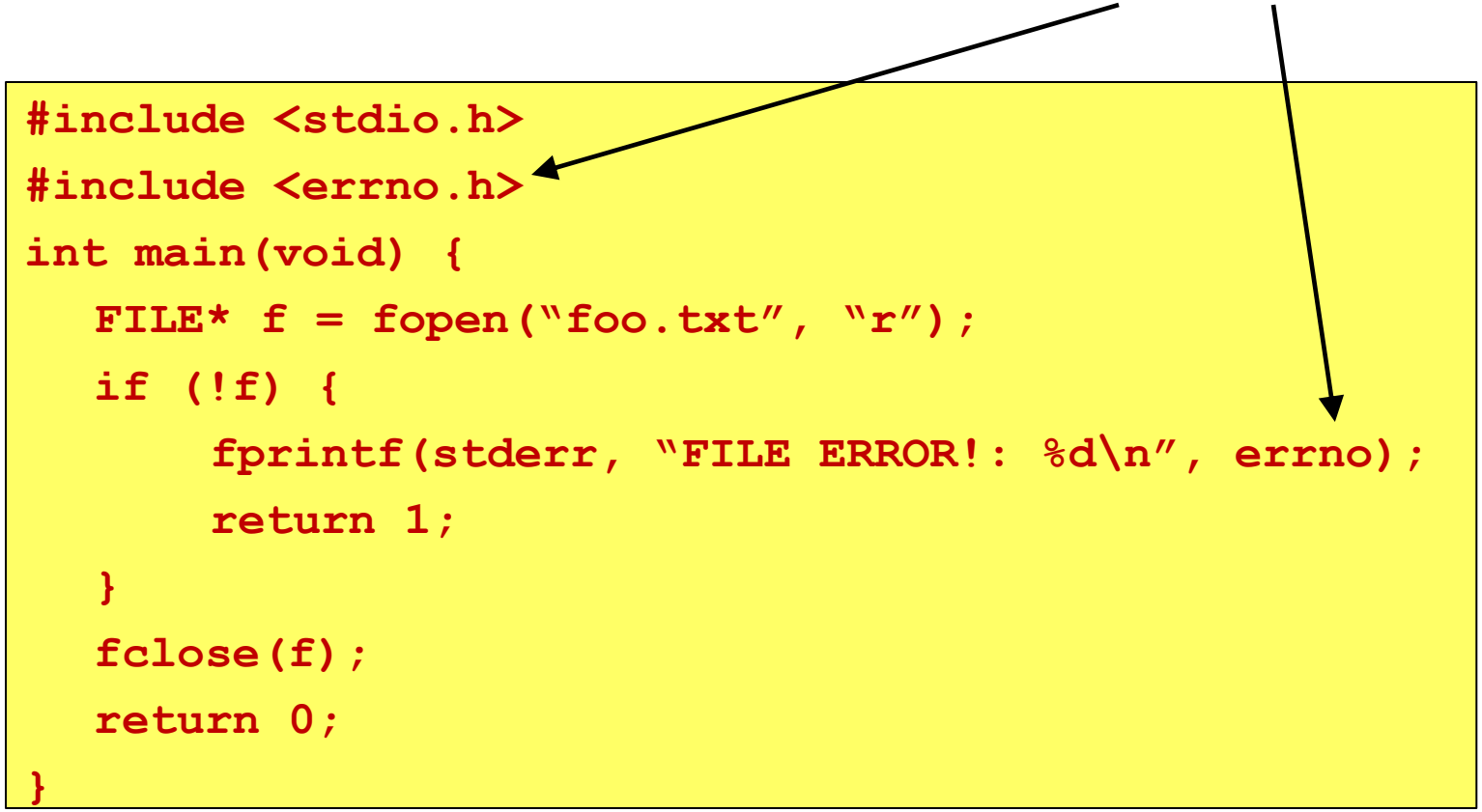
Function-level scope
(Declarations 2 & 3)

Block-level scope
(Declaration 4)

COMPILE-TIME CONCEPT

Accessing variables from different file

- Link-Time Concept *i.e.* External Linkage
- Can I access a variable from a different object file?



```
#include <stdio.h>
#include <errno.h>
int main(void) {
   FILE* f = fopen("foo.txt", "r");
   if (!f) {
        fprintf(stderr, "FILE ERROR!: %d\n", errno);
        return 1;
   }
   fclose(f);
   return 0;
}
```

Extern Variables

- errno.h header file contains:

```
extern int errno;
```

- An external variable accessible from a different object file. Scope is resolved at link-time.

- All global variables have external linkage by default

Extern Example

foo.c

```
int status = 1;

void foo(int status) {
   status = 5;
}
```

bar.c

```
extern int status;

int bar(int j) {
   return j += status;
}
```

main.c

```
int main(void) {
   foo(18);
   printf("bar(3): %d\n", bar(3));
   return EXIT_SUCCESS;
}
```

```
$ gcc -c foo.c bar.c main.c
$ gcc foo.o bar.o main.o -o main
bar(3): 4
```

Extern Functions

- All functions have external linkage by default as well

- Previous example

```
/* FUNCTION DECLARATIONS AKA PROTOTYPES AKA SIGNATURES */
extern void foo(int);
/*extern*/ int bar(int);

int main(void) {
   foo(18);
   printf("bar(3): %d\n", bar(3));

   return EXIT_SUCCESS;
}
```

Static Variables & Functions

- Opposite of `extern`

- Previously we studied static linkage of a variable in block scope.

```
void f(void) {
   /* static local variable */
   static int i;
   ……
}
```

- We can also apply static linkages in file level

Static Examples

foo.c

```
static int status = 1;
static void setStatus(int s)
{
   status = s;
}
```

bar.c

```
extern int status;
void bar(){
   status = 4;    /*link err*/
   setStatus(4); /*link err*/
}
```

- Please see variables.pdf handout from Precept 10 to see more details

Void Pointers

- Used whenever the exact type of an object is unknown or when using a generic pointer

- However, there is no such thing as a void variable.

CASTING

```
{
   void* vptr;
   int i = 10;
   vptr = &i;
   /* COMPILE-TIME ERROR */
   printf("%d\n", *vptr);
   printf("%d\n", *(int*)vptr);
}
```

```
{
   void* vptr;
   int i = 10;
   double d = 2.4;
   vptr = &i;
   /* BUG */
   d = *(double*)vptr;
}
```

Complex Pointers

- `int (*x)()`
  - x is a pointer to a function returning an integer
- `int *x()`
  - x is a function returning an integer pointer
- `int (*x)[]`
  - x is a pointer to an array of integers
- `int *x[]`
  - x is an array of integer pointers
- `int (*x[])()`
  - x is an array of function pointers… all returning integers
- `int *(*x)[]`
  - x is a pointer to an array of integer pointers

FUNCTION & ARRAY SYMBOLS HAVE HIGHER PRECEDENCE THAN POINTER SYMBOL

Example

```
int add(int a, int b);
int sub(int a, int b);

int main(void)
{
   int (*foo[2])(int, int);
   foo[0] = add;
   foo[1] = sub;

   printf("foo[0](1,2) = %d\n", foo[0](1,2));
   printf("foo[1](4,3) = %d\n", foo[1](4,3));
}
```

- OUTPUT –

foo[0](1,2) = 3
foo[1](4,3) = 1

Typedef

- To shorten long declaration statements
- Rename existing long data types

```
struct CoordinateStruct {
      int x;
      int y;
};
typedef struct CoordinateStruct Coordinate;
struct CoordinateStruct c1 = {0, 0};
Coordinate c2 = {0, 0};
```

More intelligent use of typedefs

- Using the same example

```
typedef struct CoordinateStruct {
      int x;
      int y;
} Coordinate;
Coordinate c2 = {0, 0};
/*************************************/
typedef enum BOOLEAN {FALSE, TRUE} BOOL;
BOOL flag = FALSE;
/*************************************/
```

Last Example - Typedef

- Last Example

```
typedef struct {
      int x;
      int y;
} Coordinate;
Coordinate c2 = {0, 0};
/*************************************/
typedef enum {FALSE, TRUE} BOOL;
BOOL flag = FALSE;
```