Scoping, Storage, and Multi-File C Programs

Lecture 12
Feb 23rd 2023 | COMP 211-002 | Joshua Bakita

Welcome!

Today:

- ➔ Multi-file C Programs
- ➔ More on storage specifiers

Logistics:

- ➔ 12% of the class has already started on Assignment 3

Fun fact…

The record high for today in Chapel Hill was 82 °F in 1922

The record low? 9 °F in 1978

Multi-File C Programs

And associated difficulties…

```c
struct datum {
    char* name;
    unsigned short index;
};

int main() {
    struct datum data[5] = {
        {"One", 1},
        {"Five", 5},
        {"Zero", 0},
        {"One Thousand", 1000},
        {"Fifteen", 15}
    };
    qsort(&data, 5, sizeof(struct datum), compare_numeric);
    printf("Smallest item is: %s\n", data[0].name);
    return 0;
}
```

```c
// Sort pointers to `struct datum` by `index`
// @returns <0 if a.index < b.index
//          =0 if a.index == b.index
//          >0 if a.index > b.index
int compare_numeric(const void* raw_a,
                    const void* raw_b) {
    // Interpret the `void*` as `struct datum*`
    const struct datum* a = raw_a;
    const struct datum* b = raw_b;

    return a->index - b->index;
}
```

A simple program that:

- ➔ sorts structures by index
- ➔ prints the smallest index by name

`compare_numeric()` is fairly independent. Let's put it in another file so that other programs can use it

Let's move it to `l12_compr.c`

l12_sort.c

```c
#include <stdio.h>
#include <stdlib.h>

struct datum {
    char* name;
    unsigned sho
};

int main() {
    struct datum data[5] = {
        {"One", 1},
        {"Five", 5},
```

l12_compr.c

```c
// Sort pointers to `struct datum` by `index`
// @returns <0 if a.index < b.index
//          =0 if a.index == b.index
//          >0 if a.index > b.index
int compare_num                    aw_a,
                                   aw_b) {
    // Interpret the `void*` as `struct datum*`
    const struct datum* a = raw_a;
    const struct datum* b = raw_b;

    return a->index - b->index;
}
```

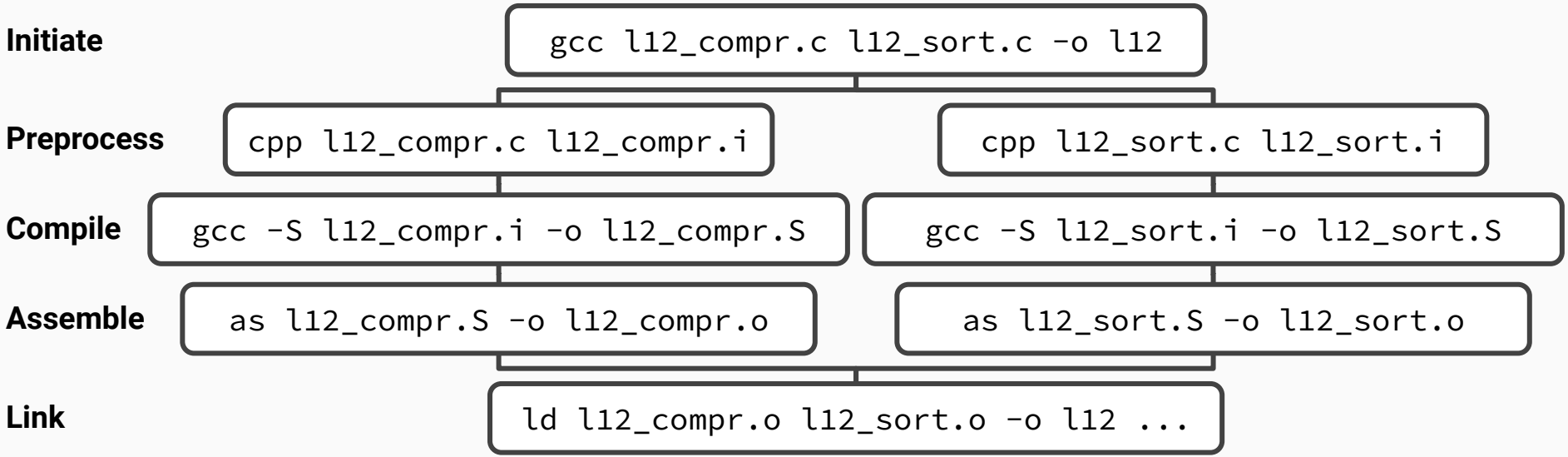
```
jbakita:COMP 211$ gcc l12_compr.c l12_sort.c -o l12
l12_compr.c: In function ‘compare_numeric’:
l12_compr.c:12:13: error: dereferencing pointer to incomplete type ‘const struct datum’
   12 |     return a->index - b->index;
      |             ^~
l12_sort.c: In function ‘main’:
l12_sort.c:17:43: error: ‘compare_numeric’ undeclared (first use in this function)
   17 |     qsort(&data, 5, sizeof(struct datum), compare_numeric);
      |                                           ^~~~~~~~~~~~~~~
l12_sort.c:17:43: note: each undeclared identifier is reported only once for each function it appears in
jbakita:COMP 211$
```

Why all the errors? It's the same code, right?

Multi-File C Programs

Behind the C Compilation Process

This process is nearly identical for C++



```c
#include <stdio.h>
#include <stdlib.h>

struct datum {
    char* name;
    unsigned sho
};

int main() {
    struct datum data[5] = {
        {"One", 1},
        {"Five", 5},
```

l12_sort.c

```c
// Sort pointers to `struct datum` by `index`
// @returns <0 if a.index < b.index
//          =0 if a.index == b.index
//          >0 if a.index > b.index
int compare_num                    aw_a,
                                   aw_b) {
    // Interpret the `void*` as `struct datum*`
    const struct datum* a = raw_a;
    const struct datum* b = raw_b;

    return a->index - b->index;
}
```

l12_compr.c

```
jbakita:COMP 211$ gcc l12_compr.c l12_sort.c -o l12
l12_compr.c: In function ‘compare_numeric’:
l12_compr.c:12:13: error: dereferencing pointer to incomplete type ‘const struct datum’
   12 |     return a->index - b->index;
      |             ^~
l12_sort.c: In function ‘main’:
l12_sort.c:17:43: error: ‘compare_numeric’ undeclared (first use in this function)
   17 |     qsort(&data, 5, sizeof(struct datum), compare_numeric);
      |                                           ^~~~~~~~~~~~~~~
l12_sort.c:17:43: note: each undeclared identifier is reported only once for each function it appears in
jbakita:COMP 211$
```

Errors while compiling only l12_compr.c

Errors while compiling only l12_sort.c

Three common types of declarations

Type Declaration
What is this type? Eg. what will be the in-memory layout of your `struct` or `union`?

Variable Declaration
What is the name and type of a variable?

Function Declaration
What is the name, argument types, and return type of a function?

What types of declarations am I missing?

https://PollEv.com/joshuabakita182

Grab these slides from the website to see the text up close.

```
jbakita:COMP 211$ gcc l12_compr.c l12_sort.c -o l12
l12_compr.c: In function ‘compare_numeric’:
l12_compr.c:12:13: error: dereferencing pointer to incomplete type ‘const struct datum’
   12 |     return a->index - b->index;
      |             ^~
l12_sort.c: In function ‘main’:
l12_sort.c:17:43: error: ‘compare_numeric’ undeclared (first use in this function)
   17 |     qsort(&data, 5, sizeof(struct datum), compare_numeric);
      |                                           ^~~~~~~~~~~~~~~
l12_sort.c:17:43: note: each undeclared identifier is reported only once for each function it appears in
jbakita:COMP 211$
```

```c
#include <stdio.h>
#include <stdlib.h>
extern int compare_numeric(const void*, const void*);

struct datum {
    char* name;
    unsigned short index;
};

int main() {
    struct datum data[5] = {
        {"One", 1},
        {"Five", 5},
        {"Zero", 0},
        {"One Thousand", 1000},
        {"Fifteen", 15}
    };
    qsort(&data, 5, sizeof(struct datum), compare_numeric);
    printf("Smallest item is: %s\n", data[0].name);
    return 0;
}
```

```c
struct datum {
    char* name;
    unsigned short index;
};

// Sort pointers to `struct datum` by `index`
// @returns <0 if a.index < b.index
//          =0 if a.index == b.index
//          >0 if a.index > b.index
int compare_numeric(const void* raw_a,
                    const void* raw_b) {
    // Interpret the `void*` as `struct datum*`
    const struct datum* a = raw_a;
    const struct datum* b = raw_b;

    return a->index - b->index;
}
```

Try it yourself!

```
$ wget https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l12/l12_compr.c
$ wget https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l12/l12_sort.c
$ gcc l12_compr.c l12_sort.c -o l12
$ ./l12
```

```c
#include <stdio.h>
#include <stdlib.h>
extern int compare_numeric(const void*, const void*);

struct datum {
    char* name;
    unsigned short index;
};

int main() {
    struct datum data[5] = {
        {"One", 1},
        {"Five", 5},
        {"Zero", 0},
        {"One Thousand", 1000},
        {"Fifteen", 15}
    };
    qsort(&data, 5, sizeof(struct datum), compare_numeric);
    printf("Smallest item is: %s\n", data[0].name);
    return 0;
}
```

```c
struct datum {
    char* name;
    unsigned short index;
};

// Sort pointers to `struct datum` by `index`
// @returns <0 if a.index < b.index
//          =0 if a.index == b.index
//          >0 if a.index > b.index
int compare_numeric(const void* raw_a,
                    const void* raw_b) {
    // Interpret the `void*` as `struct datum*`
    const struct datum* a = raw_a;
    const struct datum* b = raw_b;

    return a->index - b->index;
}
```

As these lines can be shared verbatim, try putting them in `l12_shared.h` and `#include` that file in both programs.

This makes your code more concise!

The `static` keyword

And its unfortunate dual meanings…

The `static` keyword

From before...

Static Memory
Global variables, `static` variables, string literals.

Stack Memory
Temporary variables for each function on the stack.

Heap Memory
Not used by default. Accessible via `malloc()`/`free()`-like functions

Meaning #1: Store variable in static memory

When `static` is used on a variable *inside a function*, like in:

```
static int my_lucky_num = 7;
```

That tells the compiler to put this variable in *static* memory, rather than treating it as an *automatic* variable put on the stack. Note that automatic variables:

```
auto int my_lucky_num = 7;
```

are the default, and equivalent to:

```
int my_luck_num = 7;
```

```c
#include <stdio.h>

int next_num() {
    static int i = 0;
    return i++;
}

int main() {
    printf("First next_num(): %d\n", next_num());
    printf("Second next_num(): %d\n", next_num());
    printf("Third next_num(): %d\n", next_num());
    return 0;
}
```

What will this print?

https://PollEv.com/joshuabakita182

Grab these slides from the website to see the text up close.

Try it yourself!

```
$ wget https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l12/incr.c
$ gcc incr.c -o incr
$ ./incr
```

Meaning #2: This *definition* is file-local

When `static` is used on a variable *outside a function*, or on a *function definition* like in:

```
static int my_add_helper(int a, int b);
```

That tells the compiler that `my_add_helper()` will only be used in this source file, and should not be made accessible to others during *linking*.

See the assigned readings for more details.

Questions?

See office hour calendar on the website for availability.

Contact:
Email: hacker@unc.edu
Twitter: @JJBakita
Web: https://cs.unc.edu/~jbakita

Old Well, University of North Carolina at Chapel Hill, Winter 2017