Pointers and Bits

Lecture 9
Feb 7th 2023 | COMP 211-002 | Joshua Bakita

Welcome!

Today:

- ➔ File I/O Recap
- ➔ Pointers as Arguments
- ➔ Bitwise operators

Logistics:

- ➔ All recordings up.
- ➔ Sample code links fixed in slide decks.
- ➔ Readings updated online to align with in-class content. Retrospectively added readings are bolded.

Fun fact…

Midterm 1 Logistics

Midterm review session **tonight** in 014 Sitterson Hall at 6:20 PM, hosted by the TA/LA staff.

For those with extended testing time:

➔ ARS has no space to accommodate late scheduling requests.

➔ If you do not have a confirmed reservation for 2 PM at ARS, please come to 314 Sitterson Hall (office hours room) to take your extended-time exam.

Come early and bring a writing implement for Thursday! Exam will start promptly at 2 PM.

Allowed outside materials:

➔ Double-sided sheet of letter paper with written or printed materials of your choice.

➔ Printed copy of *The C Programming Language* (1st, 2nd, or international editions allowed).

Provided:

➔ ASCII Table & Scratch Paper

Recap: File I/O

Recap: File I/O

`cat` completed

Recap: File I/O

Indexing files

"[T]he file pointer [`FILE*`] points to a structure that contains information about the file, such as the location of a buffer, **the current character position in the buffer**, whether the file is being read or written..." (Sec. 7.5, *K&R C*)

I.e. each time you read or write to a file, your index into the bytes of the file is changed. You can explicitly move it forward or back via `fseek()`.

See `man fseek` or Sec. B1.6 in *K&R C*.

Pointers as Function Arguments

Pointers as Func. Args.

Give it a try!

Confused? Take your best guess; we will step through what's happening in a moment.

Try it yourself!

```
$ wget https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l9/rects.c
$ gcc rects.c -o rects
$ ./rects
```

```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```

What will this print?

https://PollEv.com/joshuabakita182

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

Pointers as Func. Args.

Pass by *value* vs. by *reference*

```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```

Takes one 8-byte[1] argument, a `struct Rectangle` composed of two 4-byte integers.

Colloquially, `rect` is passed by *value*

Takes one 8-byte[2] argument, an address representing the location where a `struct Rectangle` is stored.

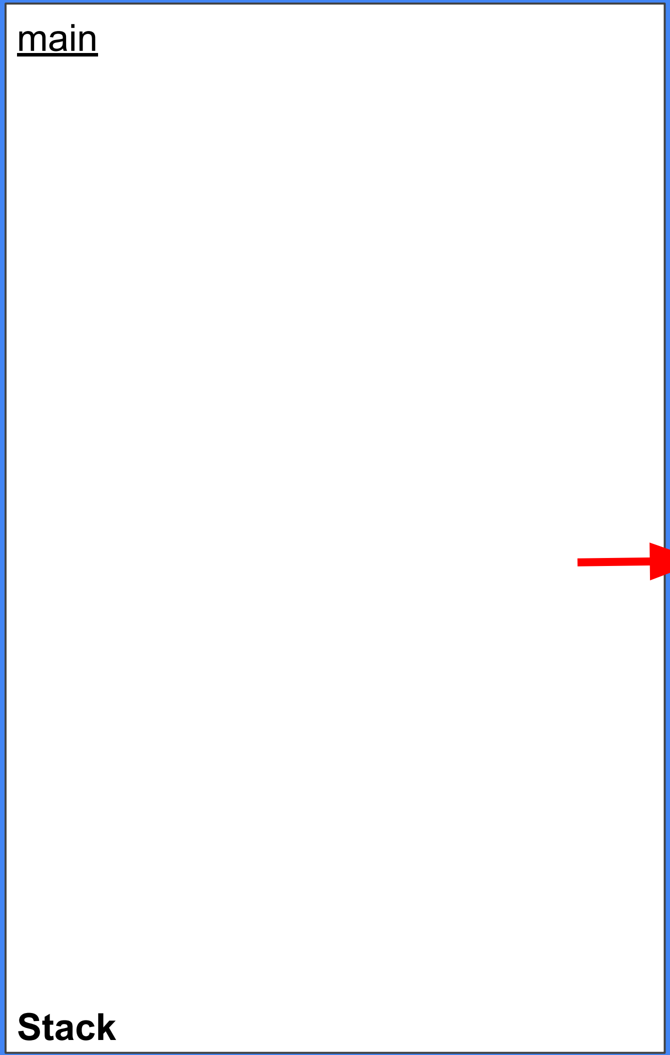
Colloquially, `rect` is passed by *reference*

Try it yourself!

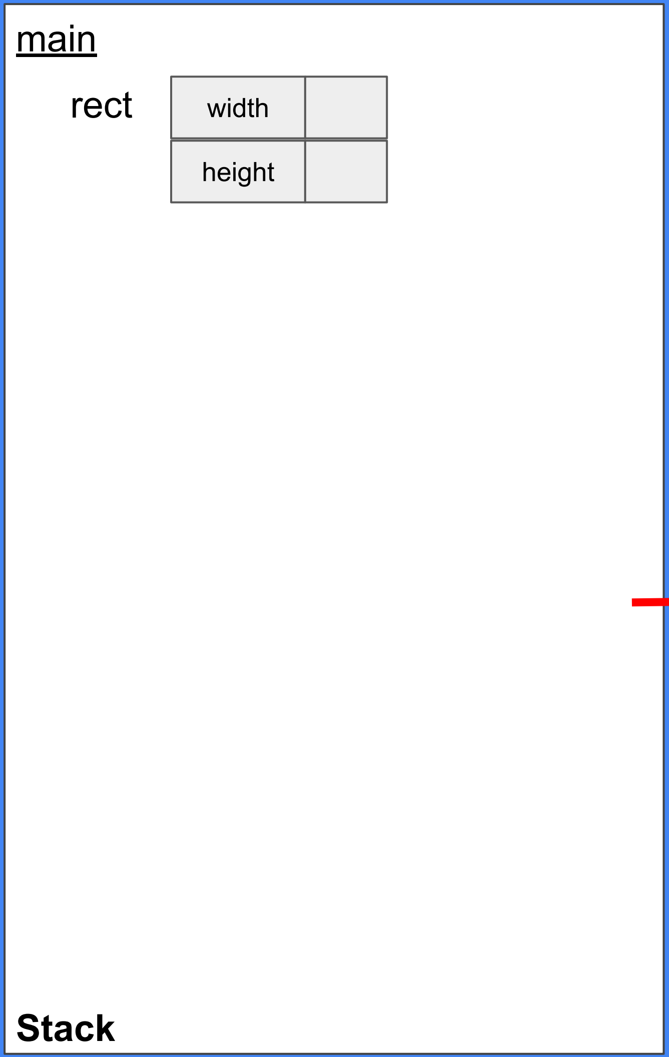
```
$ wget
https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l9/rects.c
$ gcc rects.c -o rects
$ ./rects
```

[1] *Assuming an `int` is 32-bits*

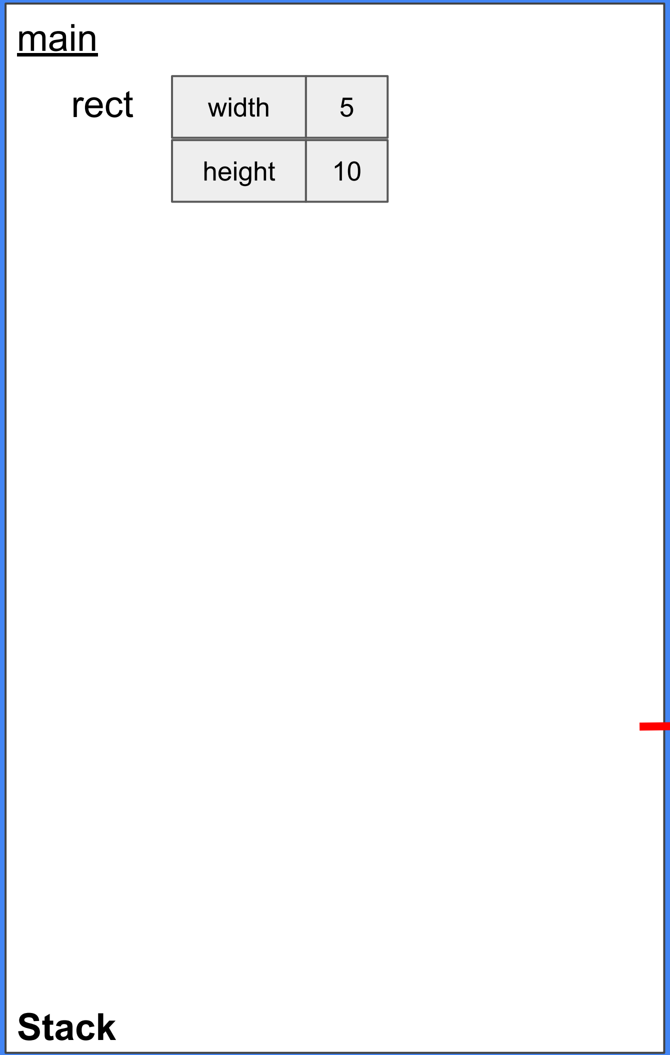
[2] *On a 64-bit system*



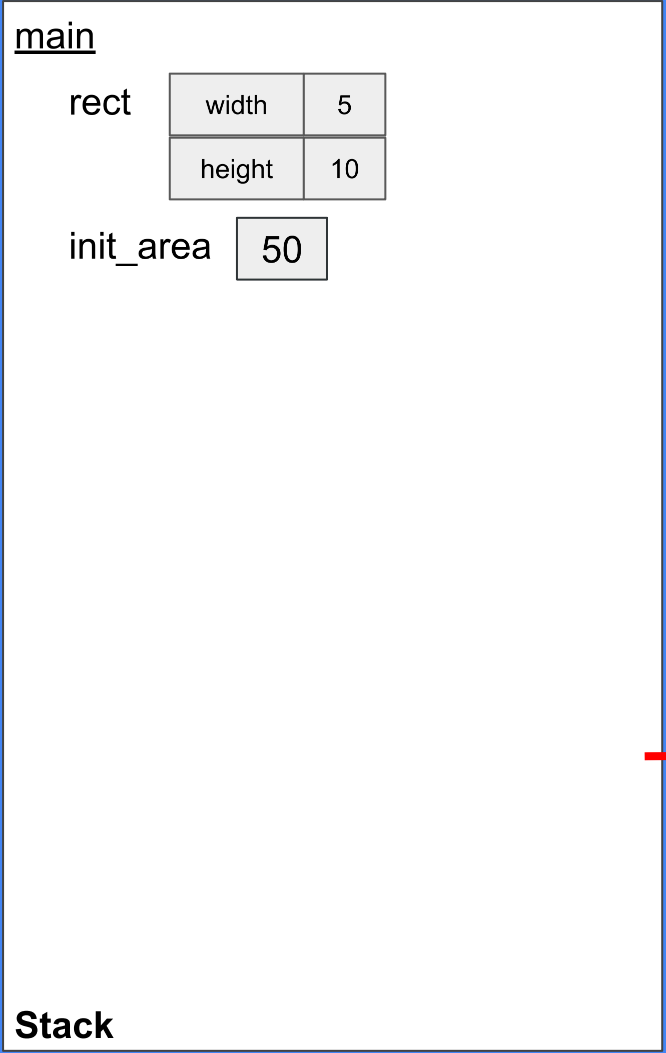
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



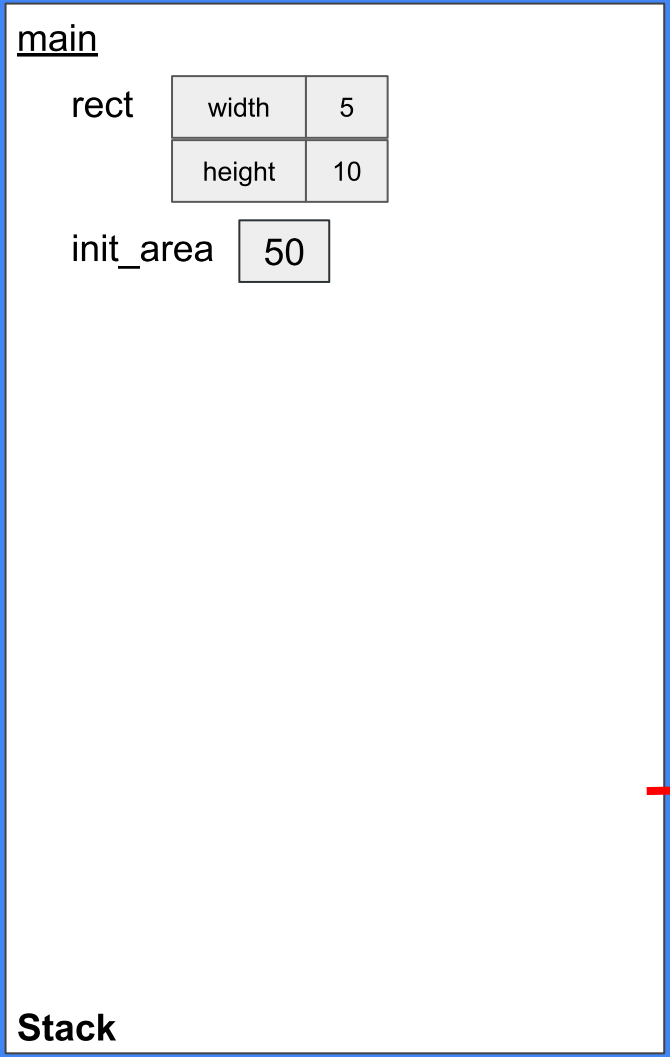
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



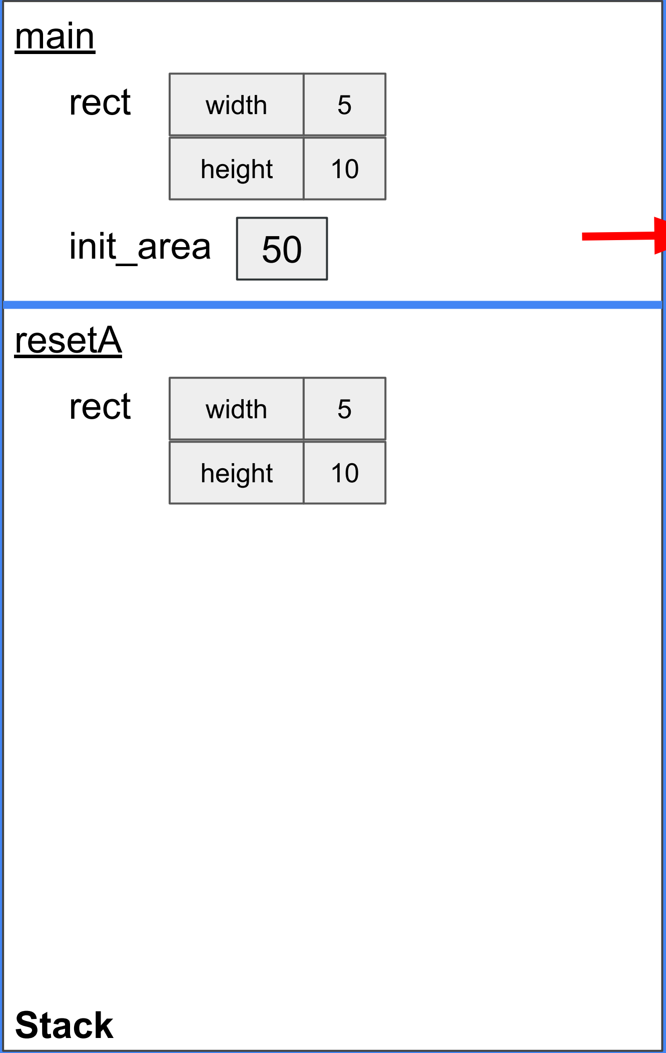
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



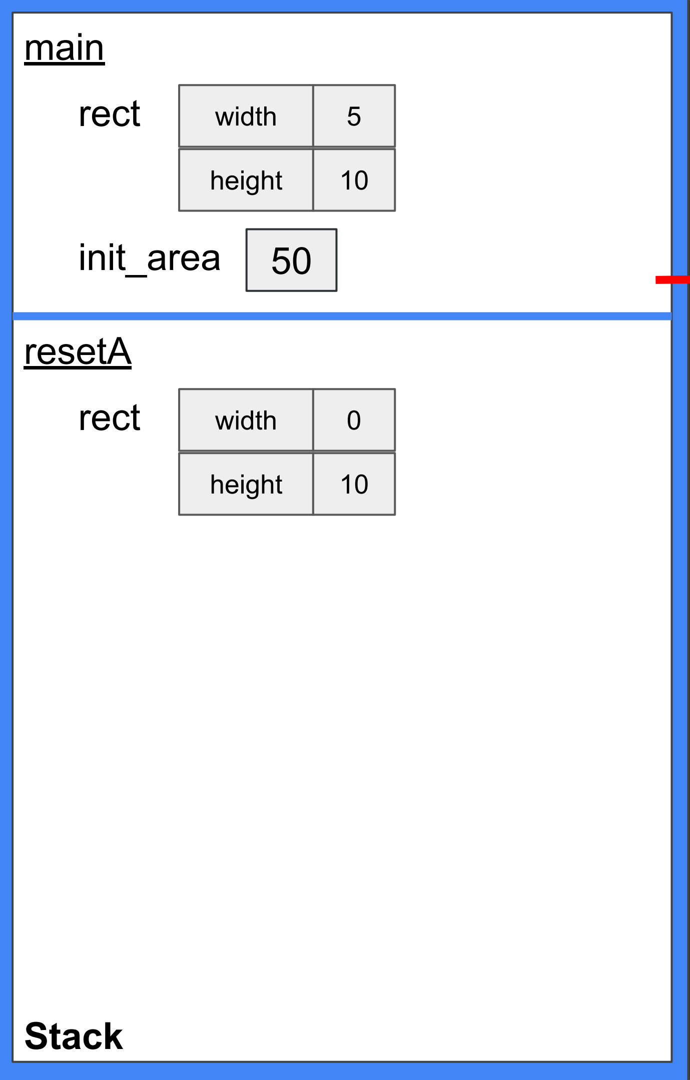
```
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



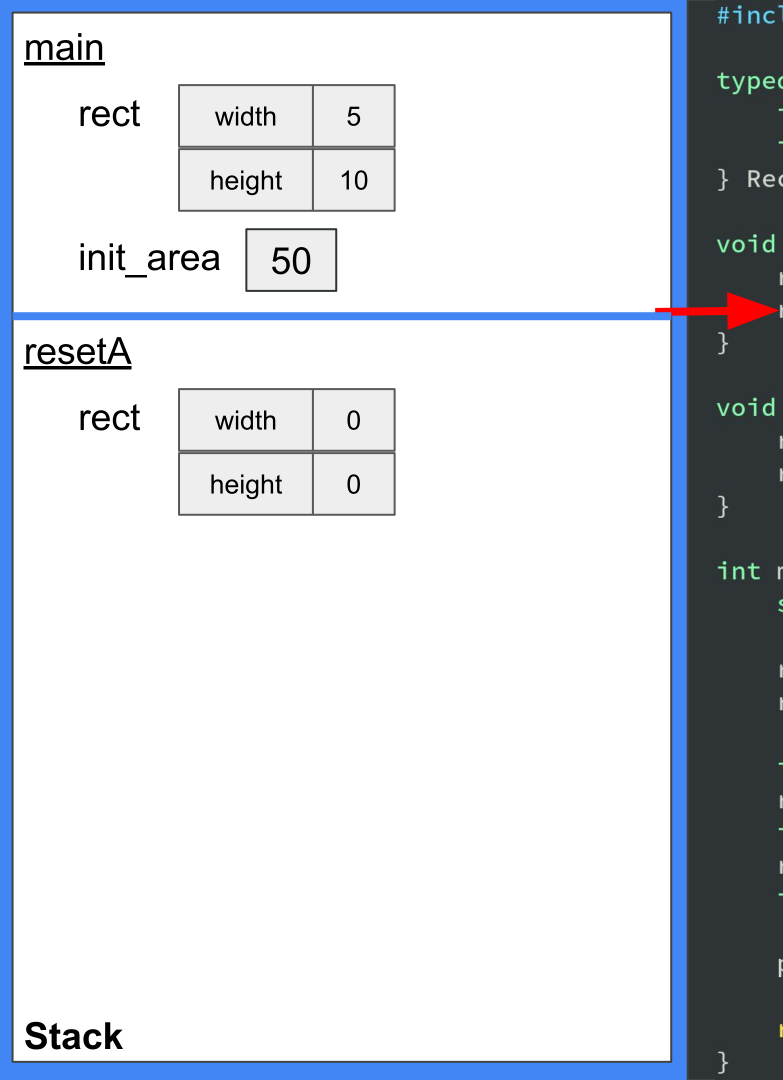
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



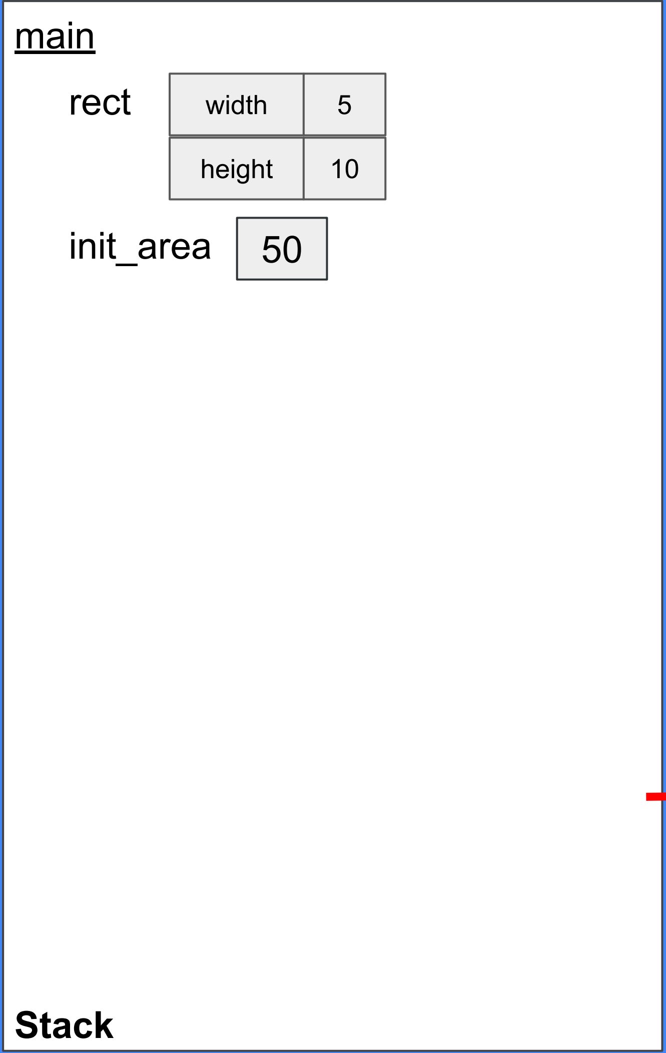
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



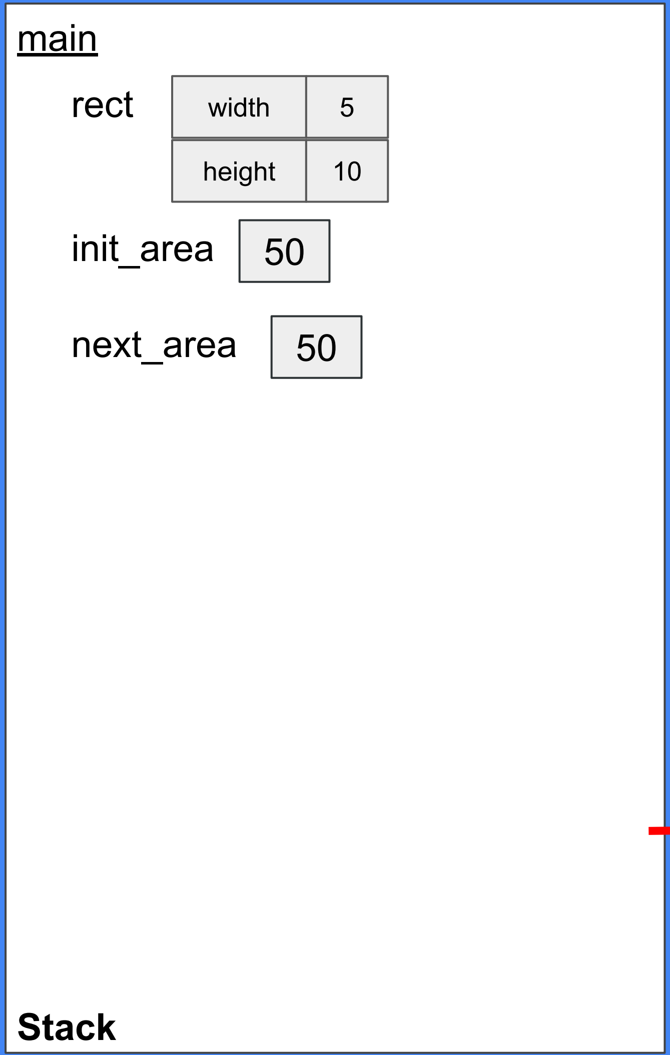
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



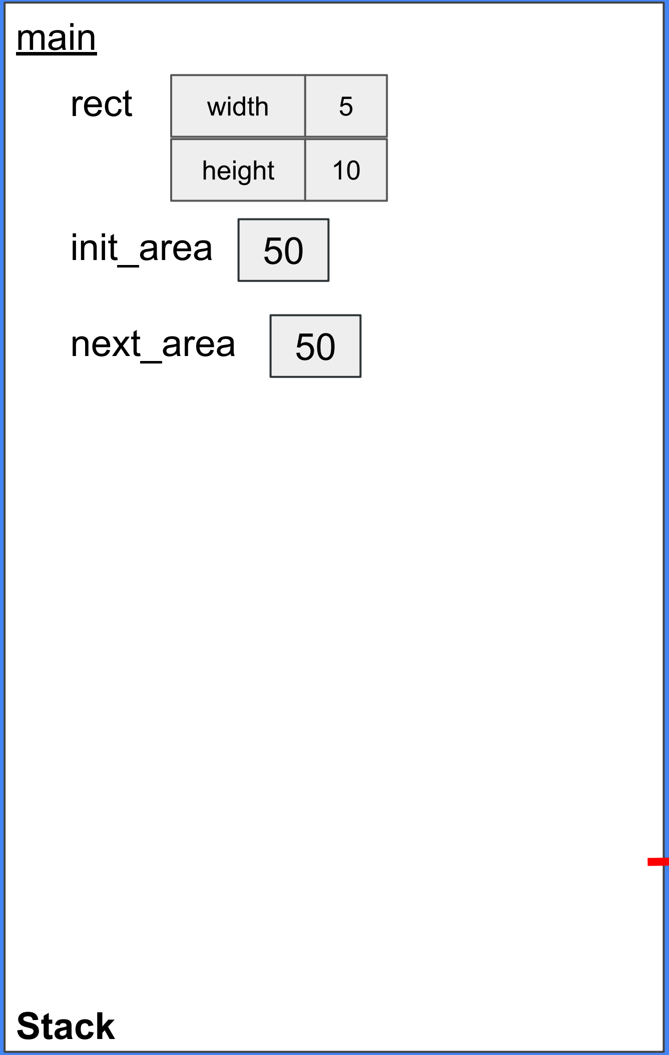
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



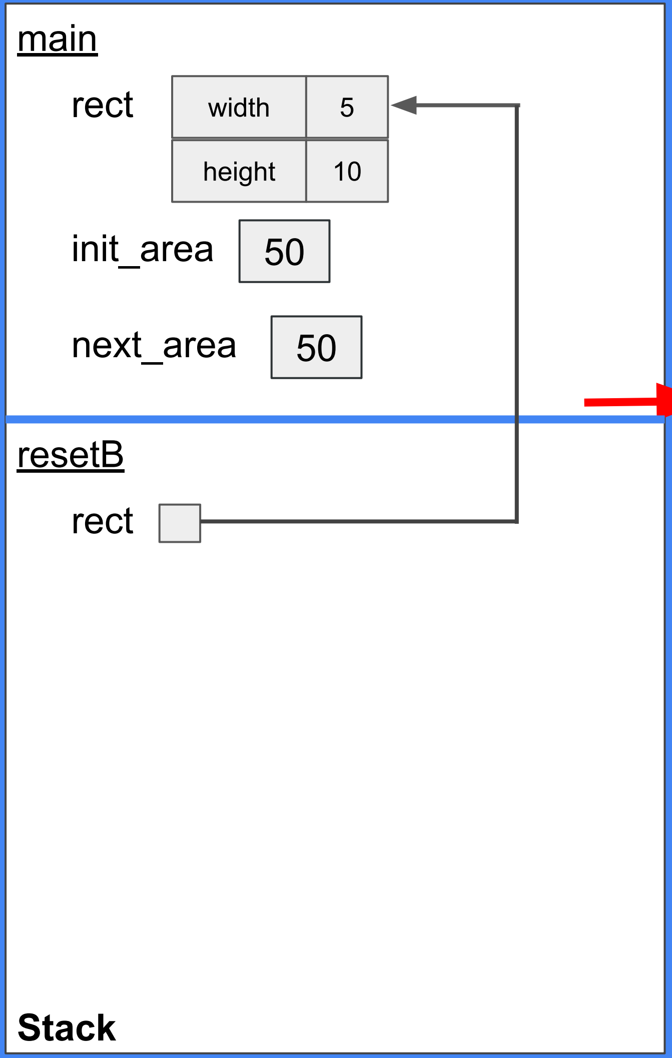
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



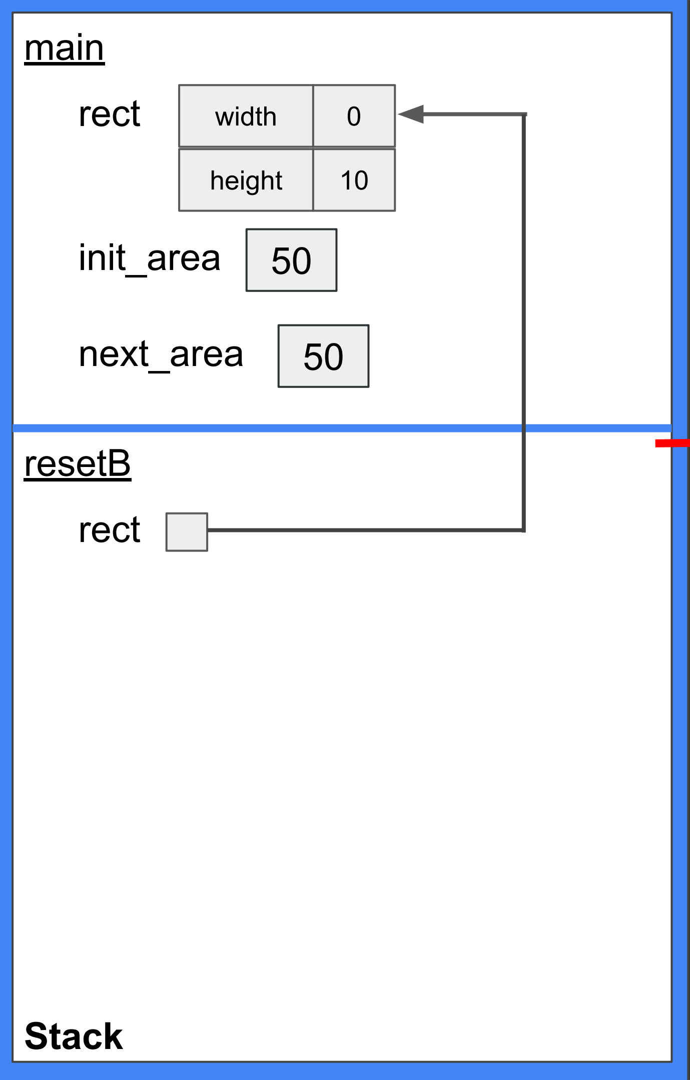
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



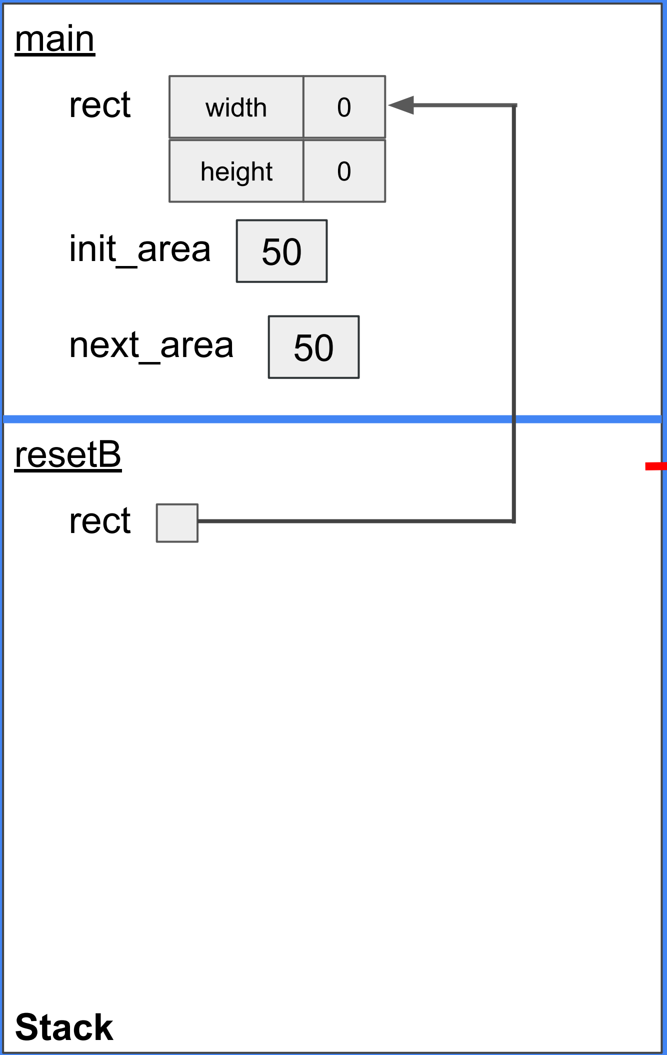
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



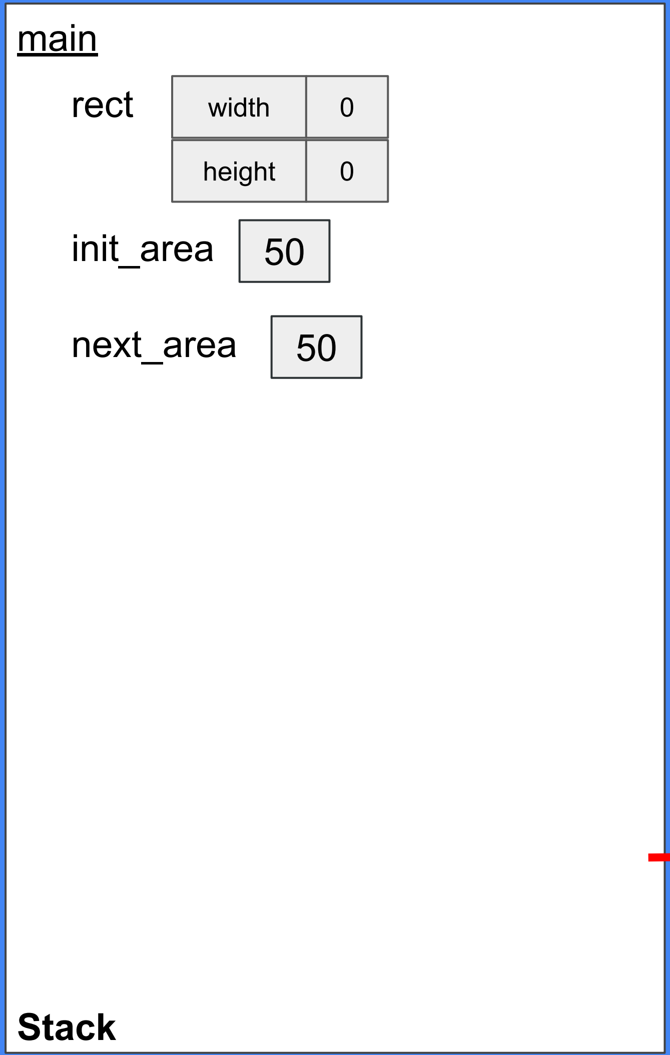
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



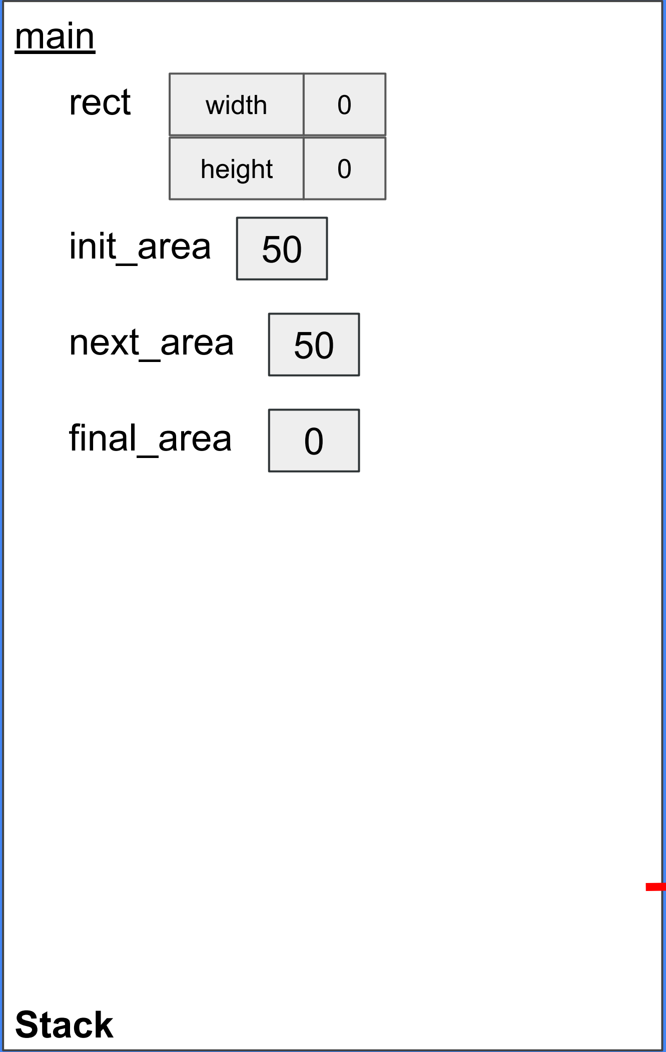
```
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



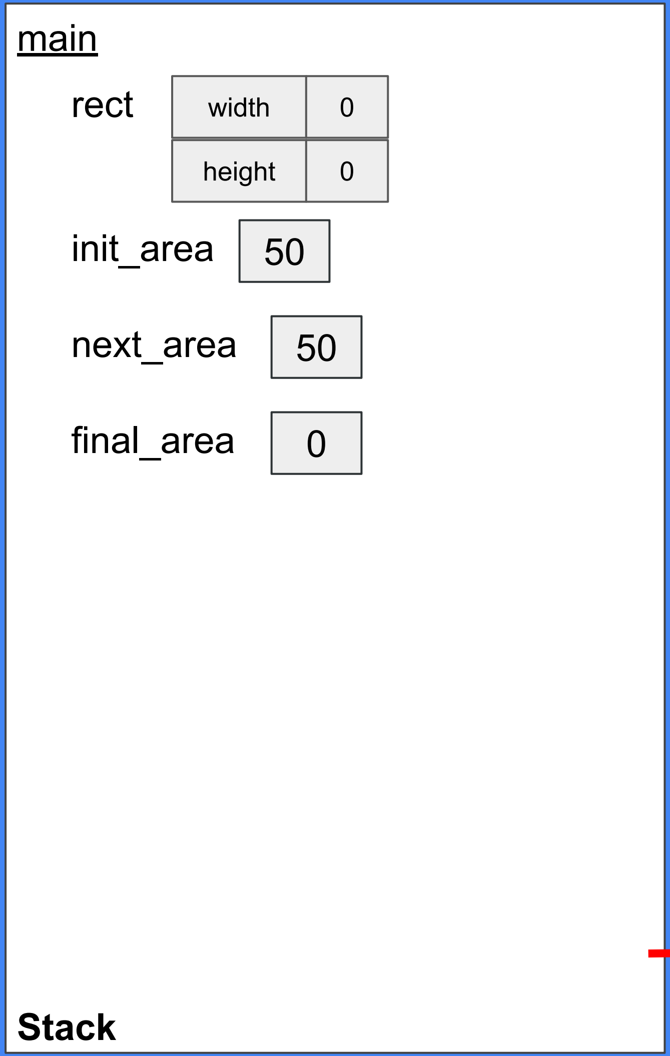
```
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```



```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```

Stack

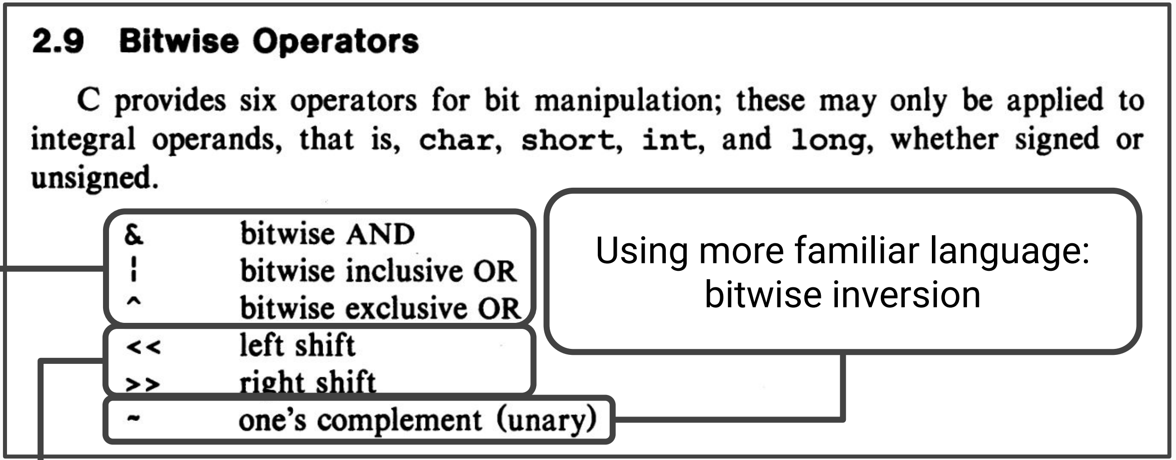
```c
#include <stdio.h>

typedef struct Rectangle{
    int width;
    int height;
} Rectangle;

void resetA(Rectangle rect) {
    rect.width = 0;
    rect.height = 0;
}

void resetB(Rectangle *rect) {
    rect->width = 0;
    rect->height = 0;
}

int main(){
    struct Rectangle rect;

    rect.width = 5;
    rect.height = 10;

    int init_area = rect.width * rect.height;
    resetA(rect);
    int next_area = rect.width * rect.height;
    resetB(&rect);
    int final_area = rect.width * rect.height;

    printf("Initially: %d, then: %d, and finally: %d\n",
           init_area, next_area, final_area);
    return 0;
}
```

Bitwise Operators

A more complete coverage…

Bitwise Operators

What are they?



2.9 Bitwise Operators

C provides six operators for bit manipulation; these may only be applied to integral operands, that is, `char`, `short`, `int`, and `long`, whether signed or unsigned.

`&`	bitwise AND
`|`	bitwise inclusive OR
`^`	bitwise exclusive OR
`<<`	left shift
`>>`	right shift
`~`	one's complement (unary)

Sec. 2.9 of *The C Programming Language*

Using more familiar language: bitwise inversion

Based on the AND, OR, and XOR logical operators you saw in COMP 283/MATH 381.

We discussed using these to adjust powers of 2 in Lecture 3

Let's give them a try!

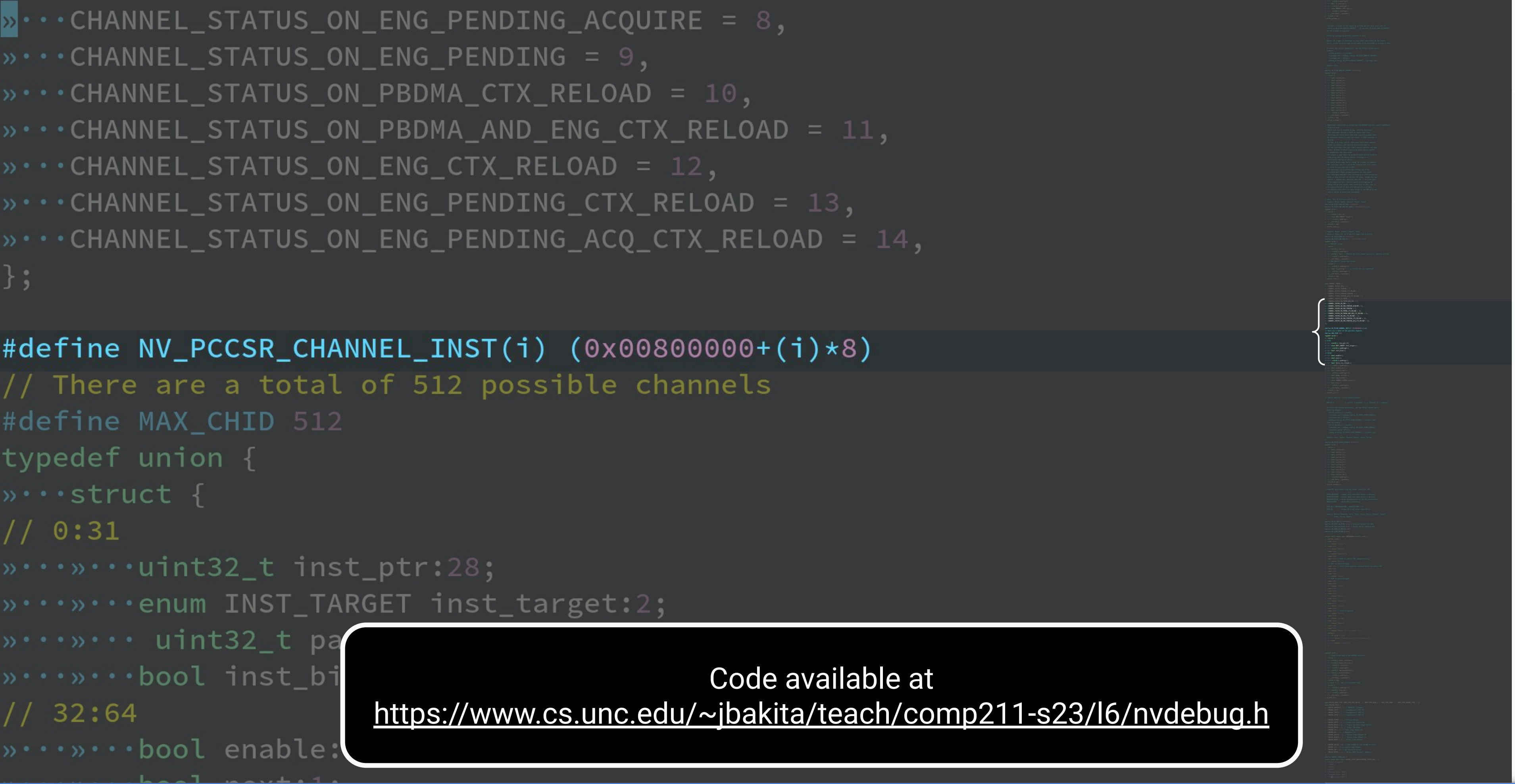
More on the Preprocessor

Beyond `#define` and `#include`

More on the Preprocessor

A few other directives

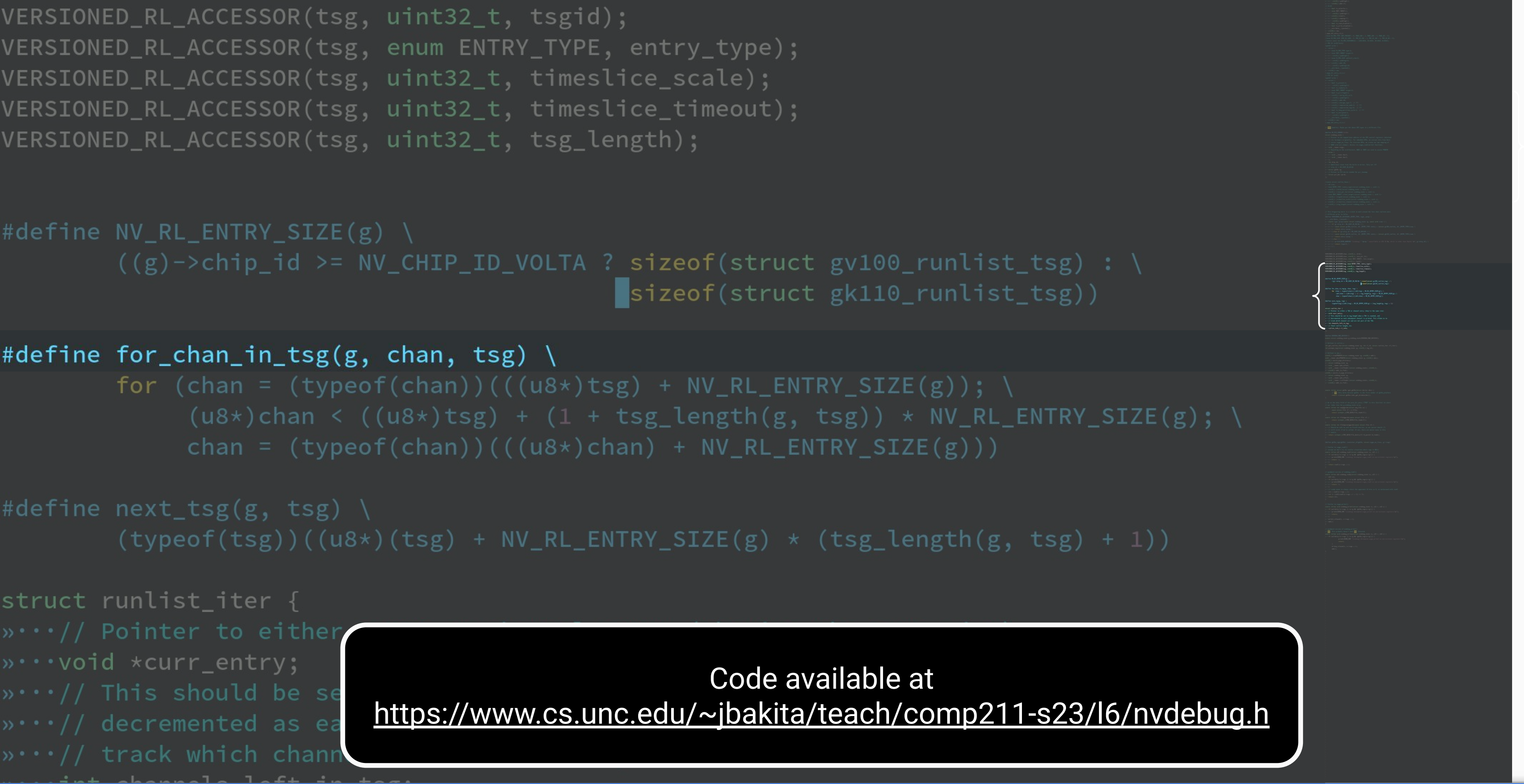
Remember: Preprocessor directives start with a #



```
»···CHANNEL_STATUS_ON_ENG_PENDING_ACQUIRE = 8,
»···CHANNEL_STATUS_ON_ENG_PENDING = 9,
»···CHANNEL_STATUS_ON_PBDMA_CTX_RELOAD = 10,
»···CHANNEL_STATUS_ON_PBDMA_AND_ENG_CTX_RELOAD = 11,
»···CHANNEL_STATUS_ON_ENG_CTX_RELOAD = 12,
»···CHANNEL_STATUS_ON_ENG_PENDING_CTX_RELOAD = 13,
»···CHANNEL_STATUS_ON_ENG_PENDING_ACQ_CTX_RELOAD = 14,
};

#define NV_PCCSR_CHANNEL_INST(i) (0x00800000+(i)*8)
// There are a total of 512 possible channels
#define MAX_CHID 512
typedef union {
»···struct {
// 0:31
»···»···uint32_t inst_ptr:28;
»···»···enum INST_TARGET inst_target:2;
»···»··· uint32_t pa
»···»···bool inst_bi
// 32:64
»···»···bool enable:
»···»···bool next:1;
```

Code available at
https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l6/nvdebug.h

nvdebug.h: More Complex #define



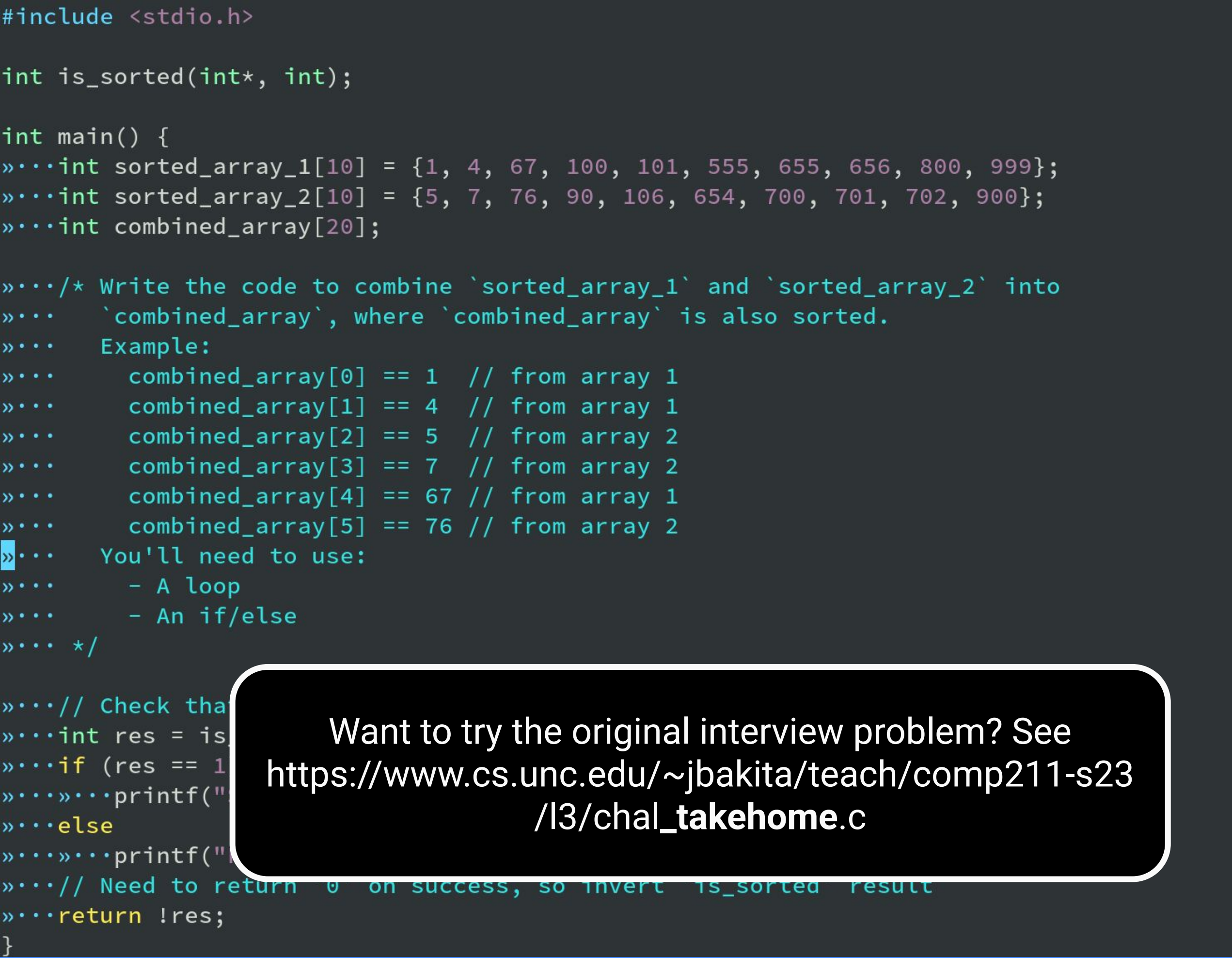
```
VERSIONED_RL_ACCESSOR(tsg, uint32_t, tsgid);
VERSIONED_RL_ACCESSOR(tsg, enum ENTRY_TYPE, entry_type);
VERSIONED_RL_ACCESSOR(tsg, uint32_t, timeslice_scale);
VERSIONED_RL_ACCESSOR(tsg, uint32_t, timeslice_timeout);
VERSIONED_RL_ACCESSOR(tsg, uint32_t, tsg_length);


#define NV_RL_ENTRY_SIZE(g) \
        ((g)->chip_id >= NV_CHIP_ID_VOLTA ? sizeof(struct gv100_runlist_tsg) : \
                                           sizeof(struct gk110_runlist_tsg))

#define for_chan_in_tsg(g, chan, tsg) \
        for (chan = (typeof(chan))(((u8*)tsg) + NV_RL_ENTRY_SIZE(g)); \
             (u8*)chan < ((u8*)tsg) + (1 + tsg_length(g, tsg)) * NV_RL_ENTRY_SIZE(g); \
             chan = (typeof(chan))(((u8*)chan) + NV_RL_ENTRY_SIZE(g)))

#define next_tsg(g, tsg) \
        (typeof(tsg))((u8*)(tsg) + NV_RL_ENTRY_SIZE(g) * (tsg_length(g, tsg) + 1))

struct runlist_iter {
»···// Pointer to either
»···void *curr_entry;
»···// This should be se
»···// decremented as ea
»···// track which chann
»···int channels_left_in_tsg;
```

Code available at
https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l6/nvdebug.h

nvdebug.h: More Complex #define

Challenge Problem

Adapted from *Cracking the Coding Interview, 4th Edition*



```
#include <stdio.h>

int is_sorted(int*, int);

int main() {
»···int sorted_array_1[10] = {1, 4, 67, 100, 101, 555, 655, 656, 800, 999};
»···int sorted_array_2[10] = {5, 7, 76, 90, 106, 654, 700, 701, 702, 900};
»···int combined_array[20];

»···/* Write the code to combine `sorted_array_1` and `sorted_array_2` into
»···   `combined_array`, where `combined_array` is also sorted.
»···   Example:
»···     combined_array[0] == 1  // from array 1
»···     combined_array[1] == 4  // from array 1
»···     combined_array[2] == 5  // from array 2
»···     combined_array[3] == 7  // from array 2
»···     combined_array[4] == 67 // from array 1
»···     combined_array[5] == 76 // from array 2
»···   You'll need to use:
»···     - A loop
»···     - An if/else
»··· */

»···// Check tha
»···int res = is
»···if (res == 1
»···»···printf("
»···else
»···»···printf("
»···// Need to return 0 on success, so invert is_sorted result
»···return !res;
}
```

Want to try the original interview problem? See https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l3/chal_**takehome**.c

To access online:

https://www.cs.unc.edu/~jbakita/teach/comp211-s23/l3/chal.c

Challenge Problem

Combine the Arrays

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