CS 240: Programming in C

Lecture 18: Midterm 2 Review

Announcements

- Midterm 2 is tomorrow!
  - Seating chart is on the webpage
  - Find your seat **before** arriving at the exam
  - If you want a left-handed desk, email me

Exam hints

- Some questions will ask you to write a function
  - Pay attention to the argument types!
  - Write the function signature first
  - Then re-read / think about how each of the arguments are used
- “declare a new type” == typedef

Topics

- Pointers
- malloc, calloc, free
- Stack vs. heap
- Linked lists
  - growing forward, backward
  - traversal, deletion
- Doubly-linked lists
  - prepend / insert
  - swap, remove
- Pointers to pointers
- Internal pointers
- Function pointers
- Recursion
- Binary Trees
  - traversal, insert
  - BSTs

Practice Questions

- Pointers
  - 7, 10, 12, 13, 18, 19, 20
- Dynamic Memory Allocation
  - 22, 24, 25
- Pointers to Structures
  - 26, 27
- Pointers to Pointers
  - 29, 31
- Internal Pointers
  - 32, 34
- Recursion
  - 36, 37, 38

Quiz Review

- [Lecture Quiz #2]
- [Takehome Quiz #3]

Practice Questions - Pointers

- What is printed by the following piece of code?

```
int x = 0;
int y = 0;
int *p = NULL;

p = &x;
*p = 5;
p = &y;
*p = 7;

printf("%d %d\n", x, y);
```

```
5 7
```

PURDUE UNIVERSITY

Practice Questions - Pointers

- What is printed by the following piece of code?

```
int x = 0;
int y = 0;
int *p = NULL;
int *q = NULL;

p = &x;
q = p;
*q = 7;
q = 3;

printf("%d %d\n", x, y);
```

```
7 0
```

Practice Questions - Pointers

- What is printed by the following piece of code?

```
int x = 0;
int y = 0;
int *p = NULL;
int *q = NULL;

p = &y;
q = &x;
p = 2;

printf(“%d %d\n”, x, y);
```

```
0 0
```

Practice Questions - Pointers

- Consider the following variable definitions

```
int x = 2;
int arr[10] = { 4, 5, 6, 7, 1, 2, 3, 0, 8, 9 };
int *p;
```

- Assume that p is initialized to point to one of the integers in arr. Which of the following statements are legitimate? Why or why not?

```
p = arr;          arr = p;          p = &arr[2];     p = arr[x];       p = &arr[x];
arr[x] = p;       arr[p] = x;       &arr[x] = p;     p = &arr;         x = *arr;
x = arr + x;      p = arr + x;      arr = p + x;     x = &(arr+x);     p++;
x = --p;          x = *p++;         x = (*p)++;      arr++;            x = p - arr;
x = (p>arr);      arr[*p] = *p;     *p++ = x;        p = p + 1;        arr = arr + 1;
```

Practice Questions - Pointers

- Consider the following variable definitions

```
int x = 2;
int arr[10] = { 4, 5, 6, 7, 1, 2, 3, 0, 8, 9 };
int *p;
```

- Assume that p is initialized to point to one of the integers in arr. Which of the following statements are legitimate? Why or why not?

`p = arr;`	`arr = p;`	`p = &arr[2];`	`p = arr[x];`	`p = &arr[x];`
`arr[x] = p;`	`arr[p] = x;`	`&arr[x] = p;`	`p = &arr;`	`x = *arr;`
`x = arr + x;`	`p = arr + x;`	`arr = p + x;`	`x = &(arr+x);`	`p++;`
`x = --p;`	`x = *p++;`	`x = (*p)++;`	`arr++;`	`x = p - arr;`
`x = (p>arr);`	`arr[*p] = *p;`	`*p++ = x;`	`p = p + 1;`	`arr = arr + 1;`

Practice Questions - Pointers

- Given the following definitions:

```
int arr[] = { 0, 1, 2, 3 };
int *p = arr;
```

- are the following two statements equivalent?

```
p = p + 1;                p++;
```

- Yes!

Practice Questions - Pointers

- Write a function called ‘swap’ that will accept two pointers to integers and will exchange the contents of those integer locations

```
void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Practice Questions - Pointers

- Show a call to this subroutine to exchange two variables

```
int a = 4;
int b = 7;
swap(&a, &b);
```

Practice Questions - Pointers

- Why is it necessary to pass pointers to the integers instead of just passing the integers to swap?
- Passing the integers would just pass the **values** -- not the variables themselves. To affect the variables, you need to pass their addresses

Practice Questions - Pointers

- What would happen if you called swap like this?

```
int x = 5;
swap(&x, &x);
```

- Nothing… x swaps with itself so the value doesn't change.

Practice Questions - Pointers

- Can you do this?

```
swap(&123, &456);
```

- No! 123 and 456 are not stored anywhere, so they don’t have addresses

Practice Questions - Pointers

- What does the following code print?

```
int func() {
  int array[] = { 4, 2, 9, 3, 8 };
  int *p = NULL;
  int i = 0;

  p = &array[2];
  p++;
  printf(“%d\n”, *(p++));
  *(--p) = 7;

  (*p)++;
  for (i = 0; i < (sizeof(array)/sizeof(int)); i++) {
    printf(“%d “, array[i]);
  }
}
```

```
3
4 2 9 8 8
```

Practice Questions - Pointers

- Write a subroutine called clear_it that accepts a pointer to integer and an integer that indicates the size of the space that the pointer points to. clear_it should set all of the elements that that pointer points to to zero.

```
void clear_it(int *arr, int n) {
  assert(arr);
  for (int i = 0; i < n; i++) {
    arr[i] = 0;
  }
}
```

PURDUE UNIVERSITY

Practice Questions - Pointers

- Write a subroutine called add_vectors that accepts three pointers to integer and a fourth parameter to indicate what the size of the spaces that the pointers point to. add_vectors should add the elements of the first two ‘vectors’ together and store them in the third ‘vector’. e.g., if two arrays of 10 integers, A and B, were to be added together and the result stored in an array C of the same size, the call would look like:

```
add_vectors(a, b, c, 10);
```

  and, as a result, c[5] would be the sum of a[5] and b[5]

Practice Questions - Pointers

```
void add_vectors(int *a, int *b, int *c, int n) {
  assert(a && b && c);
  for (int i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
  }
}
```

```
void add_vectors(int *a, int *b, int *c, int n) {
  assert(a && b && c);
  for (int i = 0; i < n; i++) {
    *(c + i) = *(a + i) + *(b + i);
  }
}
```

Practice Questions - *Dynamic Memory Allocation*

- Given the following definitions:

```
#include <malloc.h>

int   *pi = NULL;
float *pf = NULL;
char  *pc = NULL;
char  my_string[] = "Hello, World!";
```

- Write statements to do the following memory operations:
    - reserve space for 100 integers and assign a pointer to that space to pi
    - reserve space for 5 floats and assign a pointer to that space to pf
    - unreserve the space that pi points to
    - reserve space for enough characters to hold the string in my_string and assign a pointer to that space to pc. Copy my_string into that space.
    - free everything that hasn't been unreserved yet.

Practice Questions - *Dynamic Memory Allocation*

```
pi = malloc(100 * sizeof(int));

pf = malloc(5 * sizeof(float));

free(pi);
pi = NULL;

pc = malloc(sizeof(my_string) + 1);
strcpy(pc, my_string);

free(pf); pf = NULL;
free(pc); pc = NULL;
```

Practice Questions - Dynamic Memory Allocation

- What happens if you reserve memory and assign it to a pointer named p and then reserve more memory and assign the pointer to p? How can you refer to the first memory reservation?

- You can't! When you reassign p, you lose access to the first memory reservation.

Practice Questions - *Dynamic Memory Allocation*

- Does it make sense to free() something twice? What's a good way to prevent this from happening?

- No, you might end up freeing something else that was allocated at that same spot.
- Always set the pointer to NULL after calling free.

Practice Questions - Pointers to Structures

- Suppose p is a pointer to a structure and f is one of its fields. What is a simpler way of saying:

```
x = (*p).f;
```

```
x = p->f;
```

Practice Questions - Pointers to Structures

- Given the following declarations and definitions:

```
struct s {
  int x;
  struct s *next;
};
```

- What will the following code print?

Practice Questions - Pointers to Structures

```
struct s *p1 = NULL;
struct s *p2 = NULL;
struct s *p3 = NULL;
struct s *p4 = NULL;
struct s *p5 = NULL;

p5 = malloc(sizeof(struct s));
p5->x = 5;
p5->next = NULL;
p4 = malloc(sizeof(struct s));
p4->x = 4;
p4->next = p5;
p3 = malloc(sizeof(struct s));
p3->x = 3;
p3->next = p4;
```

```
p2 = malloc(sizeof(struct s));
p2->x = 2;
p2->next = p3;
p1 = malloc(sizeof(struct s));
p1->x = 1;
p1->next = p2;

printf("%d %d\n",
  p1->next->next->next->x,
  p2->next->x);
```

```
4 3
```

Practice Questions - Pointers to Pointers

- Write a subroutine called do_allocate that is passed a pointer to the head pointer to a list of block structures: do_allocate(struct block **).
- If the head pointer is NULL, do_allocate should allocate a new struct block and make the head pointer point to it.
- If the head is not NULL, the new struct block should be prepended to the list and the head pointer set to point to it.

Practice Questions - Pointers to Pointers

```c
void do_allocate(struct block **head) {
  assert(head);

  struct block *new_block = malloc(sizeof(struct block));
  assert(new_block);

  new_block->next = *head;
  *head = new_block;
}
```

Practice Questions - Pointers to Pointers

- Write a subroutine called my_free that will accept a pointer to a pointer to int and:
  - free the space pointed to by the pointer
  - set the pointer to NULL

```
void my_free(int **data) {
  free(*data);
  *data = NULL;
}
```

PURDUE UNIVERSITY

Practice Questions - Internal Pointers

- Given the following declaration:

```
struct employee {
  char *name;
  char *title;
  int id;
};
```

write a subroutine called create_employee that accepts two string parameters for the new name and title and one integer parameter for the ID. It should return a newly allocated employee structure with all of the fields filled in.

Practice Questions - Internal Pointers

```
struct employee *create_employee(char *name,
                                 char *title,
                                 int id) {
  struct employee *new_emp = malloc(sizeof(struct employee));
  assert(new_emp);

  new_emp->name = malloc(strlen(name) + 1);
  assert(new_emp->name);
  strcpy(new_emp->name, name);

  new_emp->title = malloc(strlen(title) + 1);
  assert(new_emp->title);
  strcpy(new_emp->title, title);

  new_emp->id = id;
  return new_emp;
}
```

Practice Questions - Internal Pointers

- Write a subroutine called fire_employee that accepts a pointer to pointer to struct employee, frees its storage and sets the pointer that points to the storage to NULL.

Practice Questions - Internal Pointers

```
void fire_employee(struct employee **emp) {
  assert(emp && *emp);

  if ((*emp)->name) {
    free((*emp)->name);
    (*emp)->name = NULL;
  }
  if ((*emp)->title) {
    free((*emp)->title);
    (*emp)->title = NULL;
  }

  free(*emp);
  *emp = NULL;
}
```

Practice Questions - Recursion

- Create a recursive function to compute the factorial function

```
int factorial(int n) {
  if (n == 0) {
    return 1;
  }
  return n * factorial(n - 1);
}
```

Practice Questions - Recursion

- Create a recursive function to compute the Nth element of the Fibonacci sequence

```
int fibonacci(int n) {
  if (n == 0)
    return 1;
  if (n == 1)
    return 1;

  return (fibonacci(n - 1) +
          fibonacci(n - 2));
}
```

Practice Questions - Recursion

- Implement a recursive list search. e.g. each function call should either return the list node that it’s looking at because it matches the search item or it should return the value from calling itself on the next item in the list.

Practice Questions - Recursion

```
struct node *search(struct node *cur, int val) {
  if (!cur)
    return NULL;

  if (cur->val == val)
    return cur;

  return search(cur->next);
}
```

Lecture Quiz #2 Review

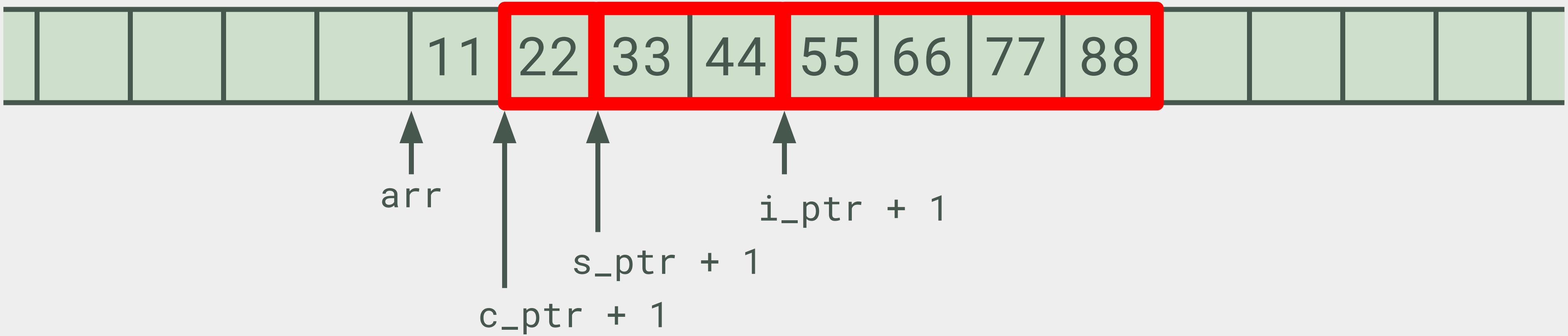
- What does this code print?

```
int main() {
  char arr[] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88 };
  char *c_ptr  = (char *)arr;
  short *s_ptr = (short *)arr;
  int *i_ptr   = (int *)arr;

  printf("%x %x %x", *(c_ptr + 1), *(s_ptr + 1), *(i_ptr + 1));
  return 0;
}
```

- Assume little endian, 64-bit system
  - char is 1 byte, short is 2 bytes, int is 4 bytes
- The expression `(short *)arr` means "cast" to a `short *`
  - Casting doesn't change the value, it only changes the pointer type

Lecture Quiz #2 Review

```
char arr[] = { 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88 };
```




```
22 4433 88776655
```

Takehome Quiz #3 Review

- Write a function to swap two nodes in a doubly-linked list
- The nodes **are not** adjacent!
  - **Extra credit** if your code *also* correctly handles the adjacent case
- Your function should look like this:

```
void swap_dbl_nodes(struct dbl_node *a,
                    struct dbl_node *b);
```

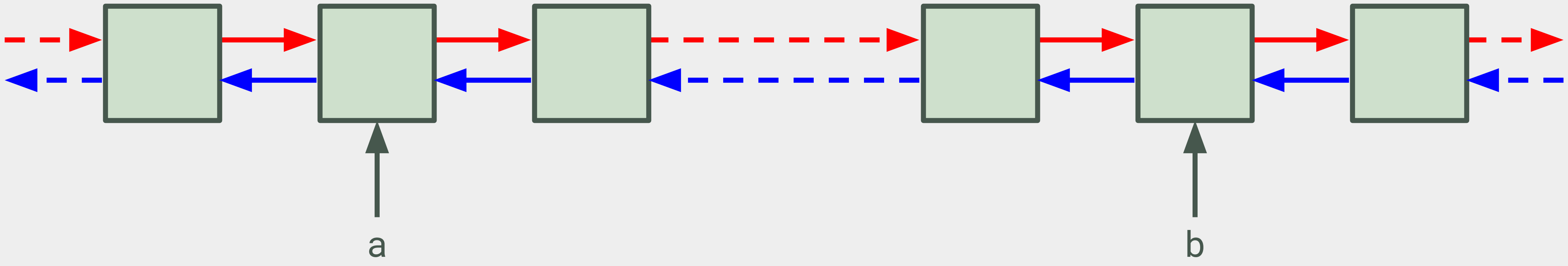
- Consider: how many steps are there?
- Hint: you may need to store temporary variables
- Assume the struct is already declared (slide 25)
- Handwritten answers ONLY
  - No need to use the template anymore
  - Limit your response to one double-sided sheet of paper

Takehome Quiz #3 Review

```
void swap_dbl_node(struct dbl_node *a, struct dbl_node *b) {
  struct dbl_node *temp = NULL;

  /* STEP 1 */
  if (a->prev != NULL)
    a->prev->next = b;

  /* STEP 2 */
  if (b->prev != NULL)
    b->prev->next = a;

  /* STEP 3 */
  if (a->next != NULL)
    a->next->prev = b;

  /* STEP 4 */
  if (b->next != NULL)
    b->next->prev = a;
```

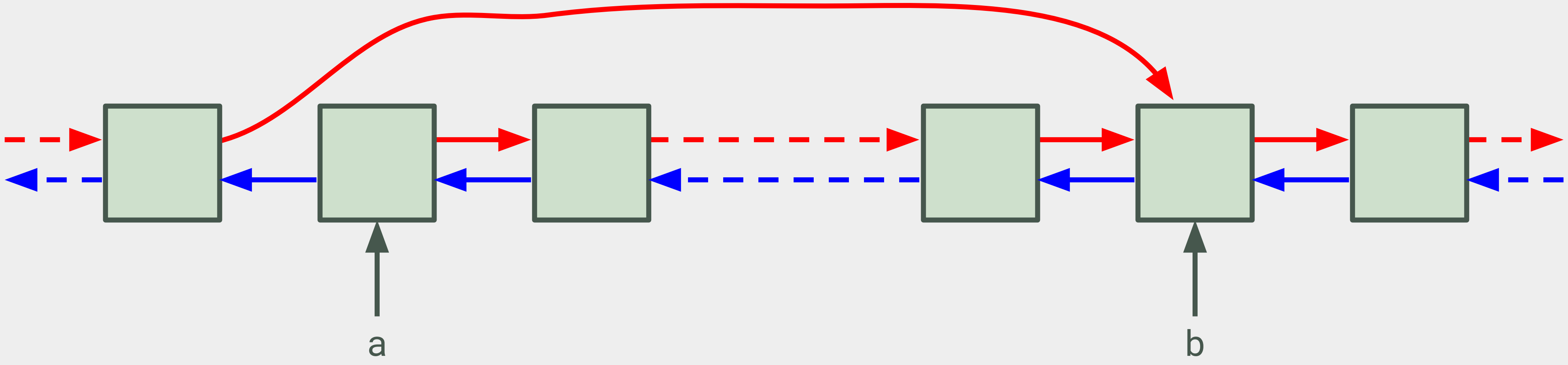
```
  temp = a->prev;
  /* STEP 5 */
  a->prev = b->prev;

  /* STEP 6 */
  b->prev = temp;

  temp = b->next;
  /* STEP 7 */
  b->next = a->next;

  /* STEP 8 */
  a->next = temp;
}
```

Takehome Quiz #3 Review



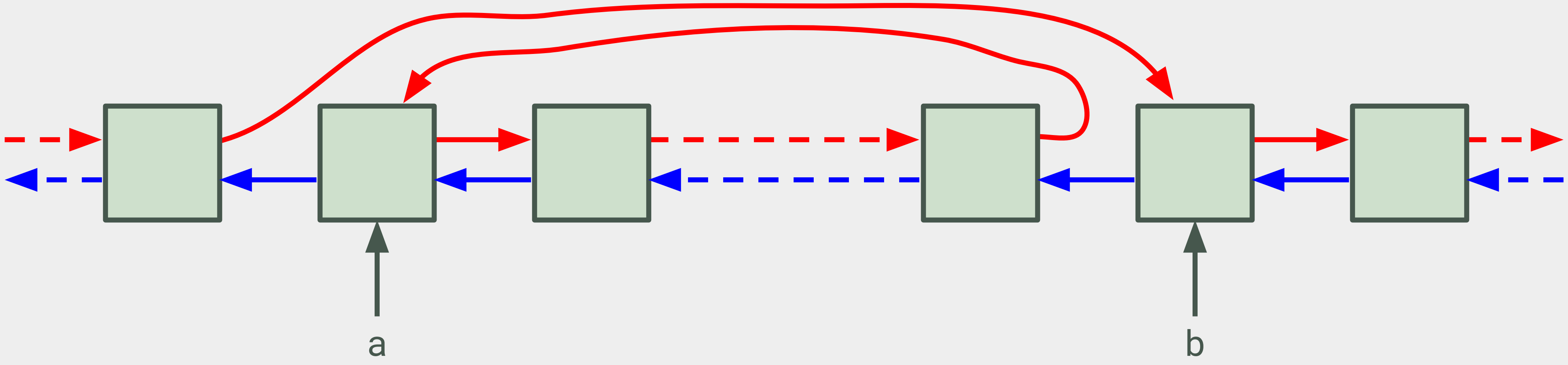
Takehome Quiz #3 Review

```
/* STEP 1 */
if (a->prev != NULL)
  a->prev->next = b;
```



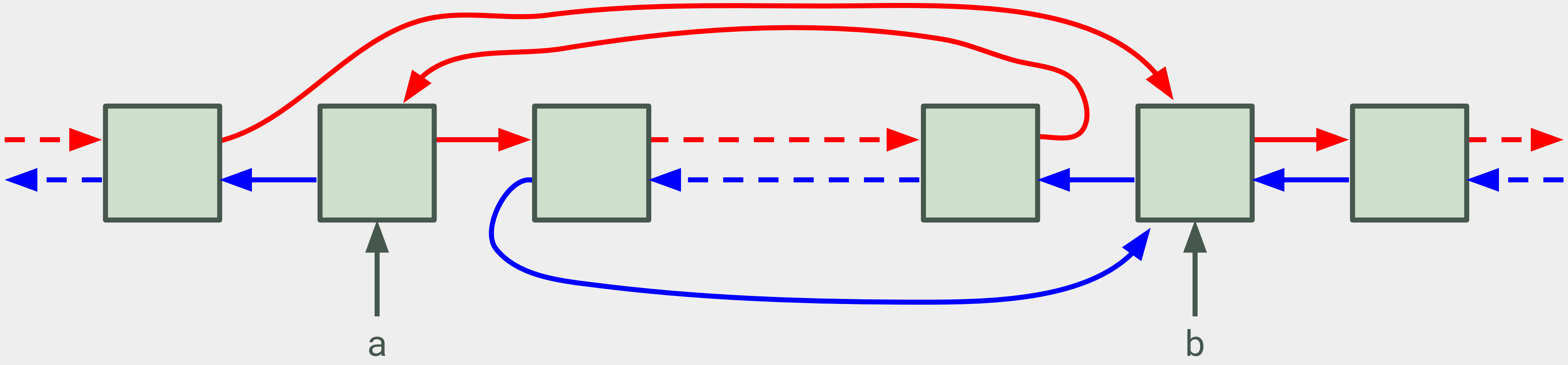
Takehome Quiz #3 Review

```
/* STEP 2 */
if (b->prev != NULL)
  b->prev->next = a;
```



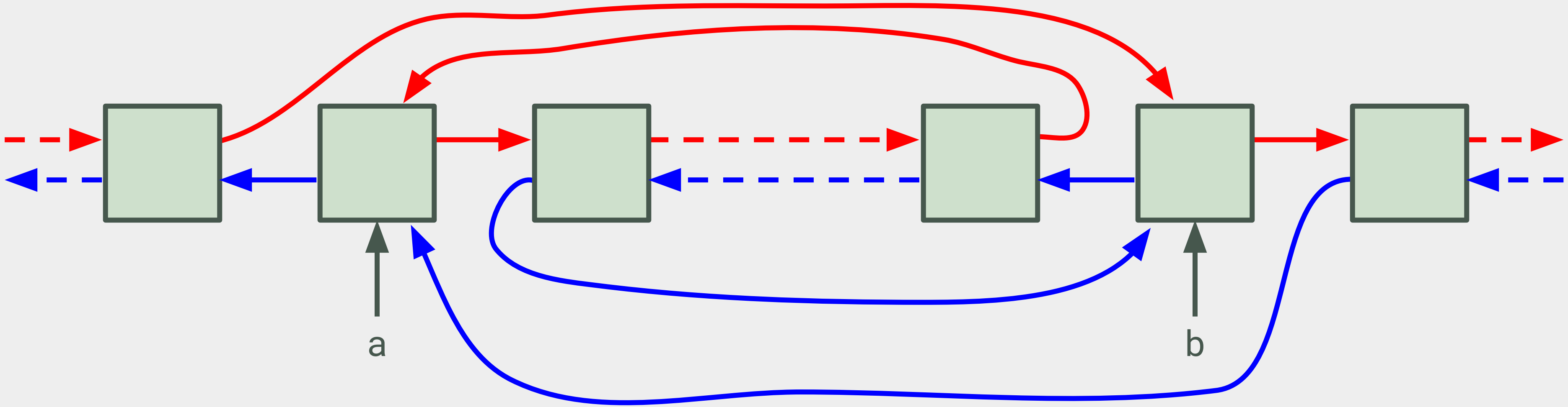
Takehome Quiz #3 Review

```
/* STEP 3 */
if (a->next != NULL)
  a->next->prev = b;
```



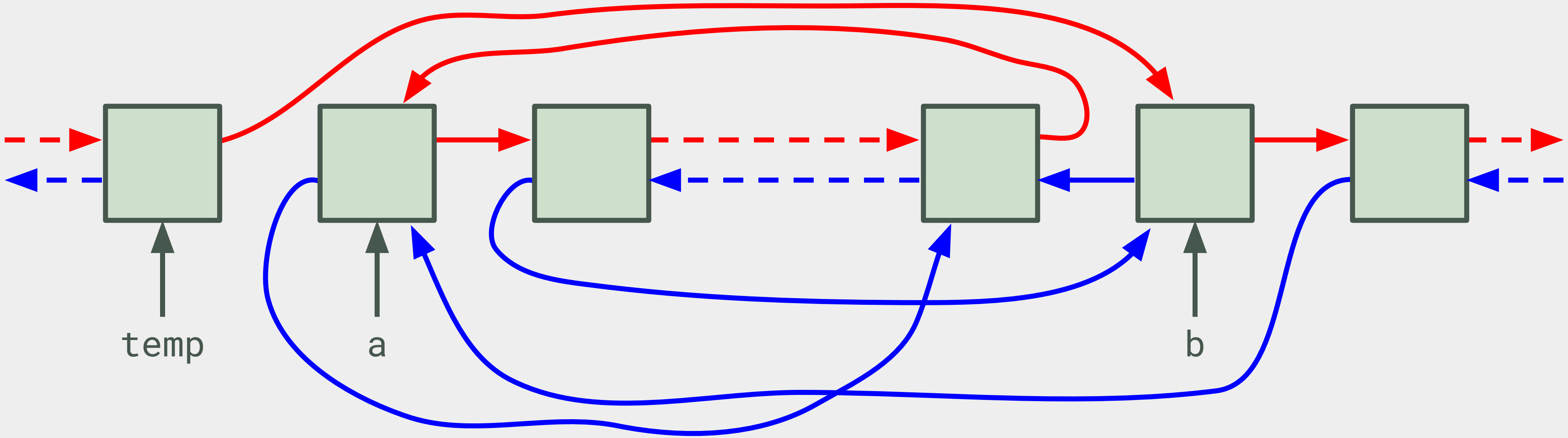
Takehome Quiz #3 Review

```
/* STEP 4 */
if (b->next != NULL)
  b->next->prev = a;
```



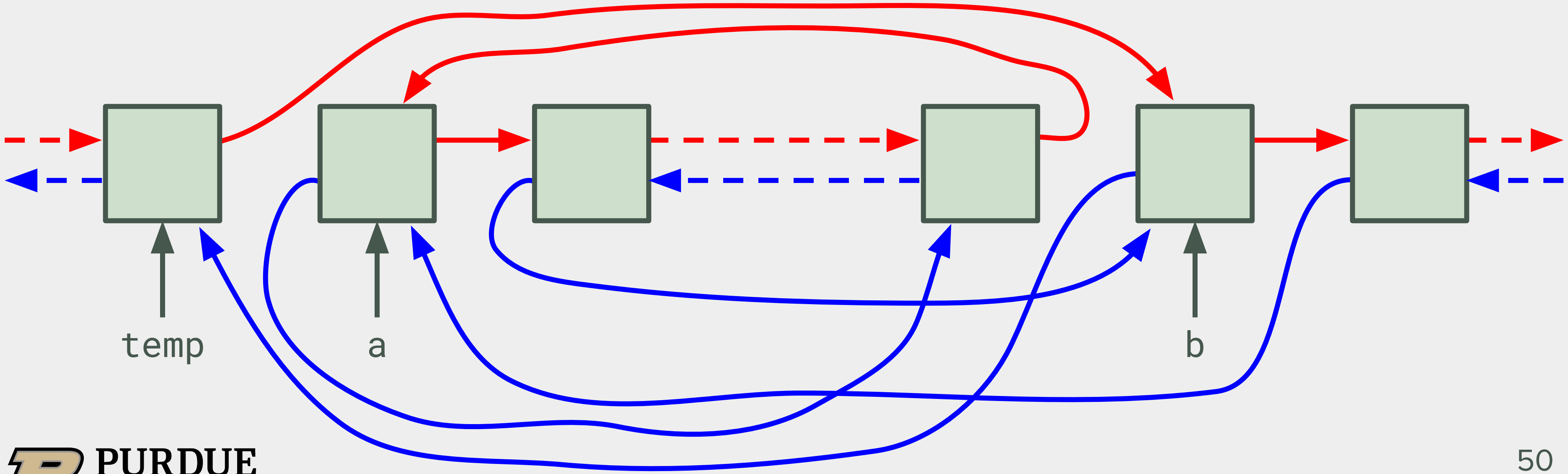
Takehome Quiz #3 Review

```
temp = a->prev;
/* STEP 5 */
a->prev = b->prev;
```



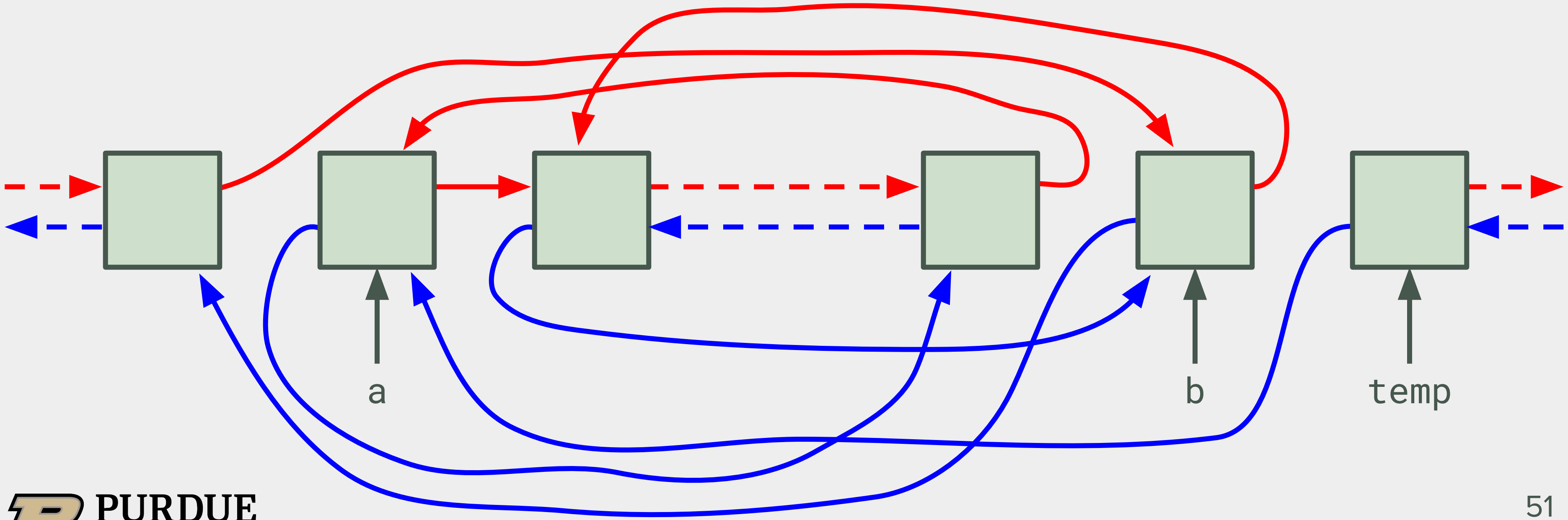
Takehome Quiz #3 Review

```
/* STEP 6 */
b->prev = temp;
```



Takehome Quiz #3 Review

```
temp = b->next;
/* STEP 7 */
b->next = a->next;
```



Takehome Quiz #3 Review

```
/* STEP 8 */
a->next = temp;
```