Welcome to COMP2521

Introductions and Icebreakers!!!

How to approach this course?

Labs

- Most labs have two marking components, a style component and a performance component
- All labs are very doable, doing these labs will be beneficial for your understanding of the content and for future assessments

Quizzes

- In my opinion, this component is the trickiest part of the course
- Do not do these quizzes last second

Assignments

- You will have two assignments during the term, the first one is usually about trees and the second one is usually about graphs
- There are no autotests in these course so make sure you test your code thoroughly

Exam

- A mix of coding and short answer questions
- In my opinion, the hardest part of the exam are the short answer questions which require a deep understanding of the data structure and algorithms taught in this course

Pointers

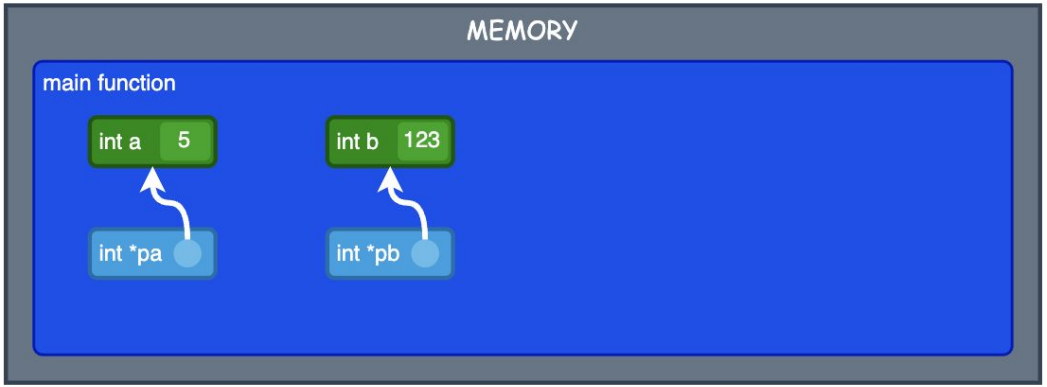
1. Using a diagram, show how the state of memory changes after each line of code is executed.

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

Pointers

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

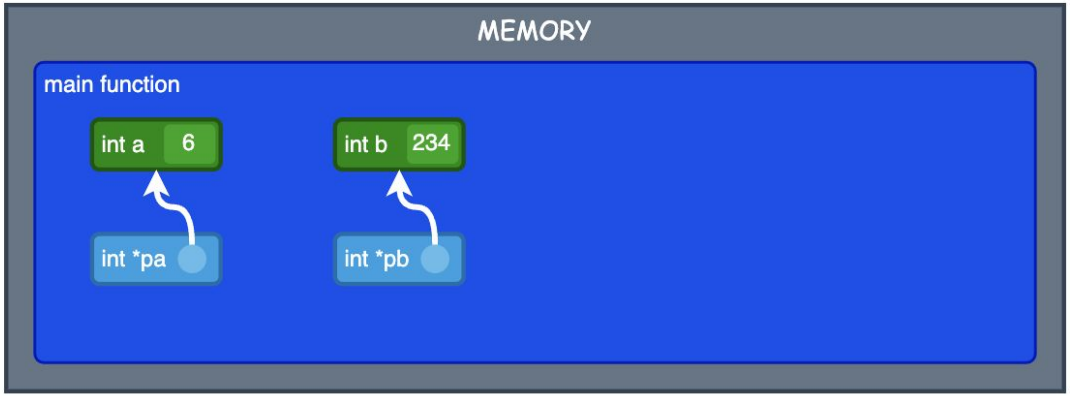
The state of memory after line 6:



Pointers

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

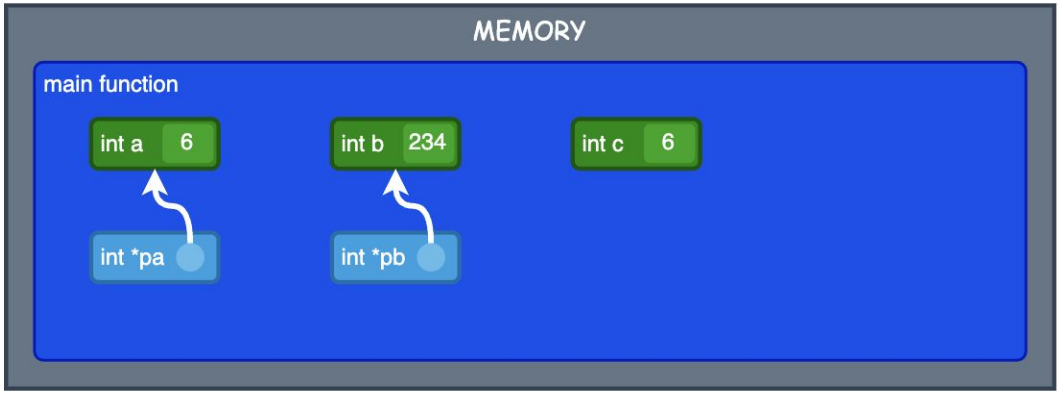
The state of memory after line 9:



Pointers

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

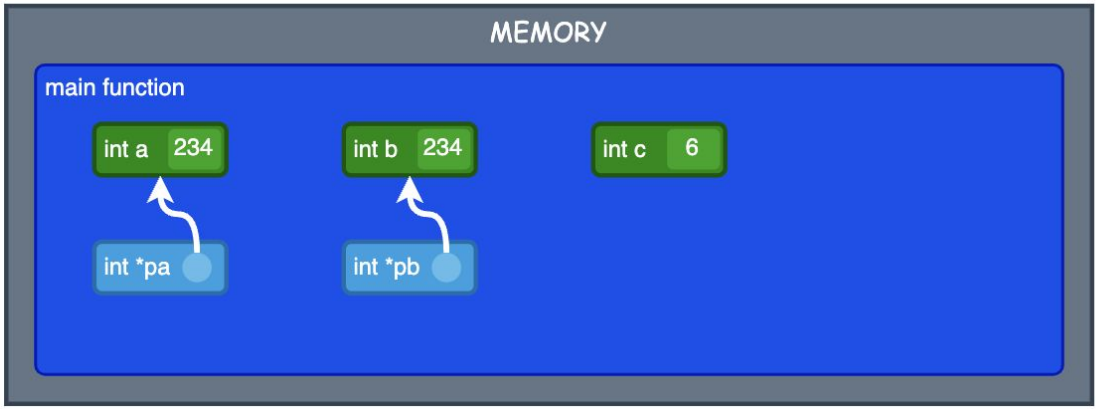
The state of memory after line 11:



Pointers

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

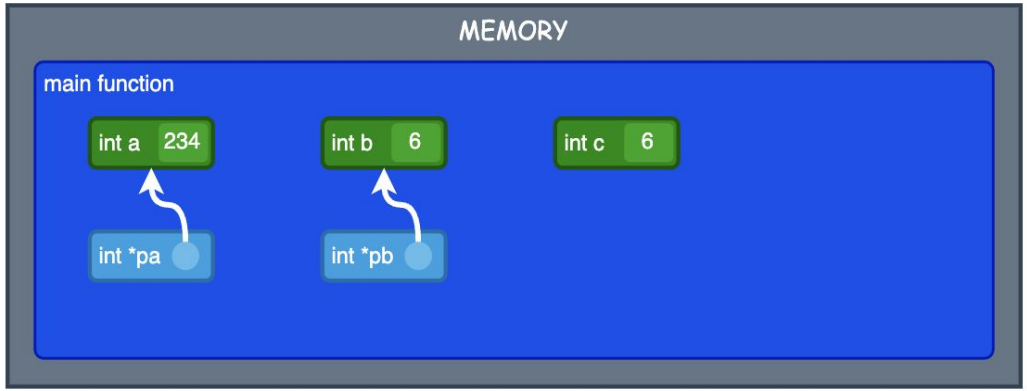
The state of memory after line 12:



Pointers

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

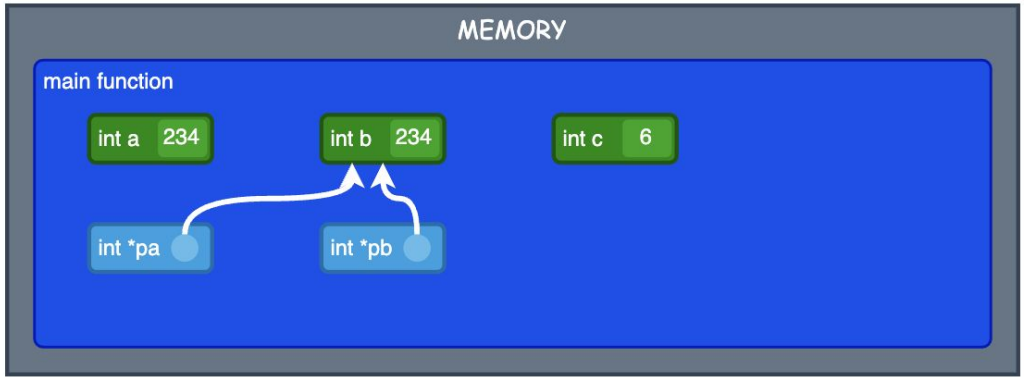
The state of memory after line 13:



Pointers

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

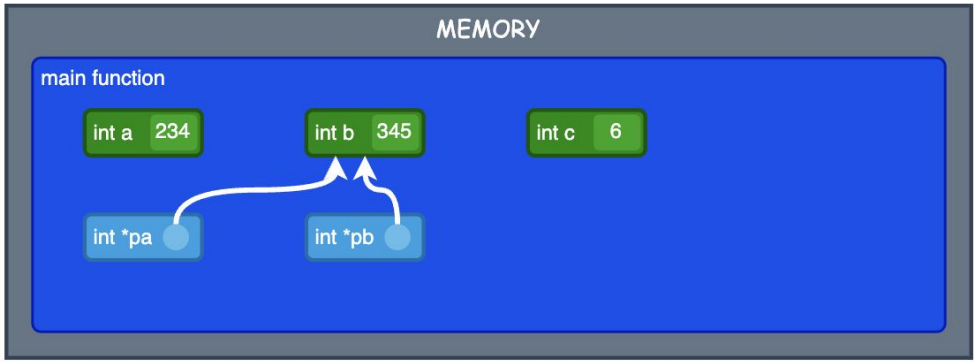
The state of memory after line 15:



Pointers

```
1  int main(void) {
2      int a = 5;
3      int b = 123;
4
5      int *pa = &a;
6      int *pb = &b;
7
8      *pa = 6;
9      *pb = 234;
10
11     int c = *pa;
12     *pa = *pb;
13     *pb = c;
14
15     pa = pb;
16     *pa = 345;
17 }
```

The state of memory after line 16:



Pointers

2. Explain why the `swap()` function here does not work as intended:

```c
int main(void) {
    int a = 5;
    int b = 7;
    swap(a, b);
    printf("a = %d, b = %d\n", a, b);
}

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

Memory Allocation

What does malloc do?

Memory Allocation

2. Explain how these two pieces of code differ:

```c
int main(void) {
    stackInt();
}

void stackInt(void) {
    int a = 5;
}
```

```c
int main(void) {
    heapInt();
}

void heapInt(void) {
    int *a = malloc(sizeof(int));
    *a = 5;
}
```

Memory Allocation

3. Modify the code below so that it allocates the struct on the heap, instead of the stack.

```c
struct node {
    int value;
    struct node *next;
};

int main(void) {
    struct node n;
    n.value = 42;
    n.next = NULL;
}
```

Memory Allocation

4. The following code creates an array of 5 integers on the stack and uses it to store some values. How can you allocate the array on the heap instead?

```c
int main(void) {
    int a[5];
    for (int i = 0; i < 5; i++) {
        a[i] = 42;
    }
}
```

Linked Lists

1. Consider the following two linked list representations:

```c
// Representation 1
struct node {
    int value;
    struct node *next;
};

int listLength(struct node *list);
```

```c
// Representation 2
struct node {
    int value;
    struct node *next;
};

struct list {
    struct node *head;
};

int listLength(struct list *list);
```

a. Compare the two representations diagramatically.
b. How is an empty list represented in each case?
c. Suppose we want to write a function that inserts a number into a list at a given position. What would the function prototype look like for each representation?
d. What are the advantages of having a separate list struct as in Representation 2?

Linked Lists

2. Consider the following simple linked list representation:

```
struct node {
    int value;
    struct node *next;
};
```

Write a function to sum the values in the list. Implement it first using `while` and then using `for`.

Linked Lists

3. Implement a function to delete the first instance of a value from a list, if it exists. Use the following list representation and prototype:

```
struct node {
    int value;
    struct node *next;
};

struct list {
    struct node *head;
};

void listDelete(struct list *l, int value);
```