Today’s plan

- Class interactions
  - Ex 3-3
  - Recap the concepts
  - Quiz
- Class exercises
  - Ex 4-1

Ex 3-3

Volunteers?

- Using GDB to trace and debug programs.

Recap - pointers

- Binky pointer fun: `https://www.youtube.com/watch?v=5VnDaHBi8dM`
- `<type> * <var>;`. a pointer of a data type. `<var>` stores an address of a variable which is a `<type>`.
    - e.g. `float *foo;`. `foo` stores an address of a variable which is a `float`.
- Two related operators:
    - Address-of operator `&`: get the "start" address of the memory that a variable is stored
    - De-referencing operator `*`: get the value stored in the memory that a a pointer is pointing to

    -
      ```
      int a = 0;
      int *b = &a;
      printf("The address is %p\n", &a);
      printf("The value is %d\n", *b);
      ```
        - `&a` gets the address of the memory where `a` is stored.
        - `*b` de-reference `b` to get the value.

Recap - pointers

- `<type> * <var>;`. More technical details:
  - `<type>` essentially tells how to interpret the address that the pointer stores.
  - Given `<type>`, we know the size. e.g. `int`, we have 4 bytes.
  - We also know how values are represented in binary (more details come later). e.g. an integer 32, we represent it as `0100000`. In 32 bits binary, it is `00000000 00000000 00000000 01000000`, or in hexadecimal, it is `0x00000020`.
  - When de-referencing, we uses the value of `<var>`, which is an **address**, and the size of `<type>`, to interpret the binary representation in the memory starting from `<var>` to `<var>` + size of `<type>`

Recap - pointers

At line 7:

Symbols	Address	Values
a	0x0000aa00	0
b	0x0000aa04	0x0000aa00 (0)

```
1       int a = 0;
2       int *b = NULL;
3       b = &a;
4       printf("The address is %p\n", &a);
5       printf("The address is %p\n", b);
6       printf("The value is %d\n", a);
7       printf("The value is %d\n", *b);
```

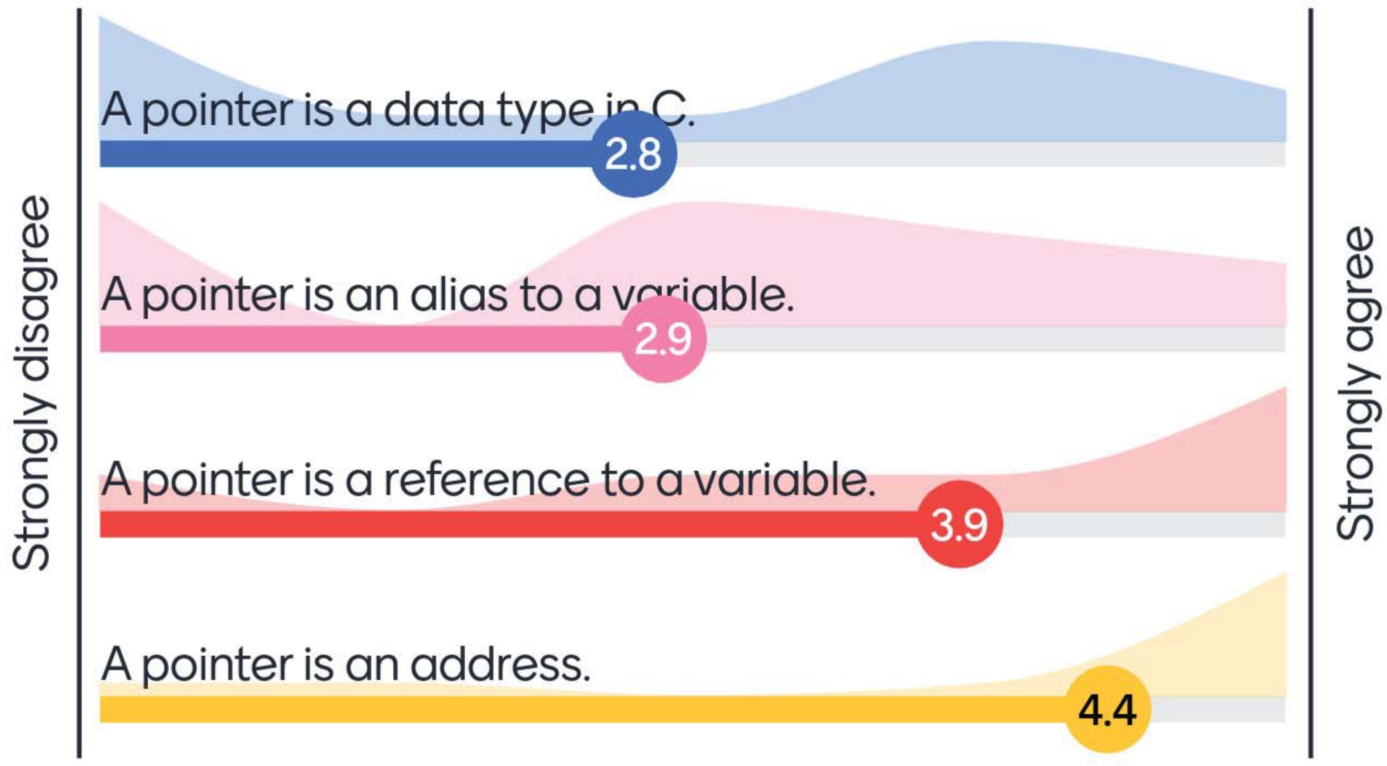
- Line 1: a is declared and defined as an `int`. In the memory, it is stored in address `0x0000aa00`.
- As it is an `int`, it has size of 4 bytes. It is initialized to 0.
- Line 2: `b` is declared and defined as a pointer of `int`. In the memory, it is stored in address `0x0000aa04`.
- Line 3: We use address-of operator to assign the address of a to `b`.
- Line 4: We use address-of operator to print the address of a.
- Line 5: We print the value of `b`, which is the address of a.
- Line 6: We print the value of a.
- Line 7: We use de-reference operator to interpret the memory starting from `0x0000aa00` to `0x0000aa04` as an `int` and print it out. That is value of a.

Quiz

Quiz!

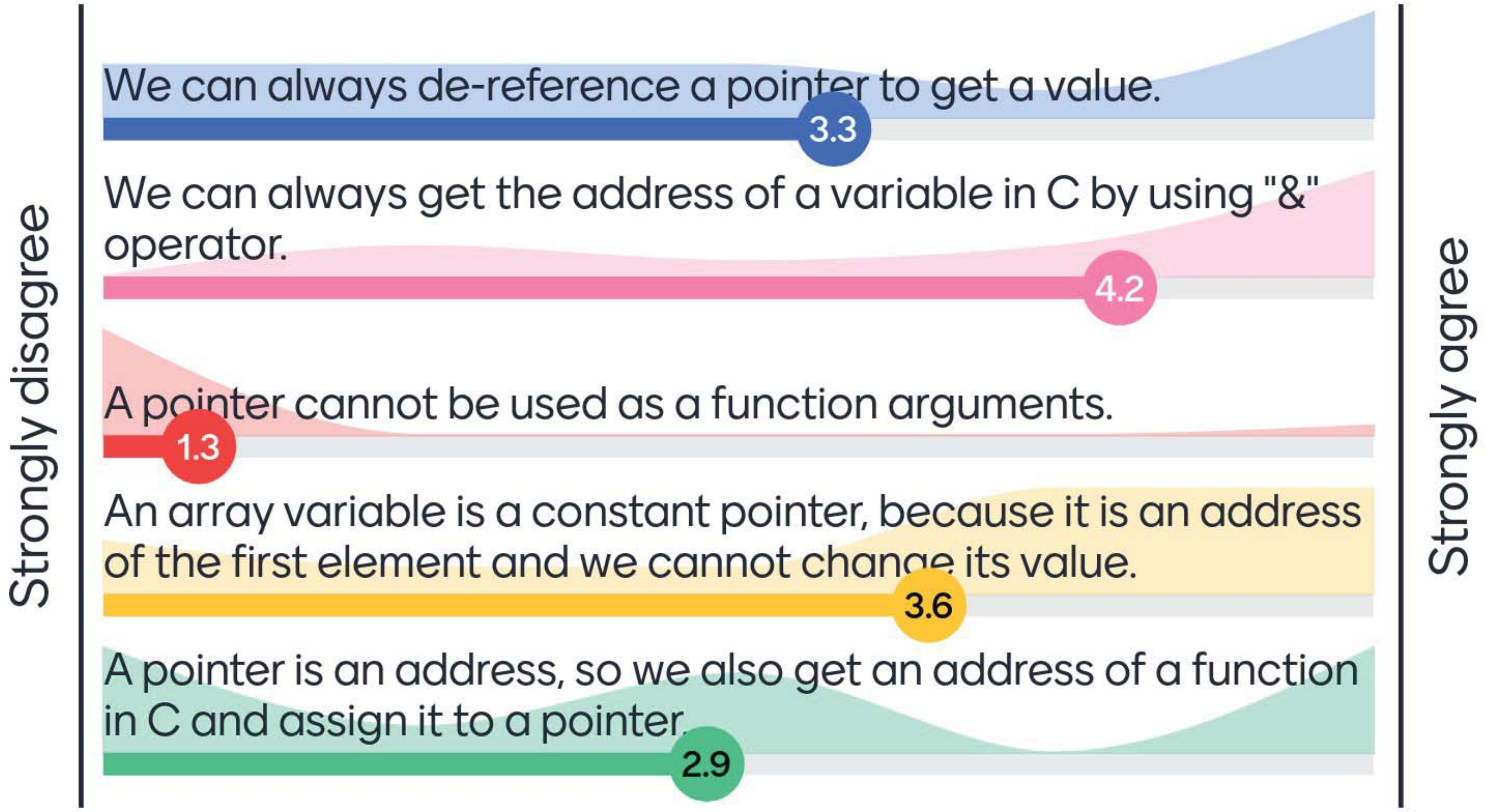
What is a pointer?

Mentimeter



13

Mentimeter



12

Mentimeter

If a is an int variable, and p is a variable whose type is pointer-to-int, how do you make p point to a?

p = &a;	✓
p = &a	✗ 7x
p=&a	✗
p=&a;	✗

The correct answer is: p = &a;

11

int a; int *p = &a; How can we use p to indirectly modify the value of a, instead of directly using a. i.e. how do we use de-referencing?

Mentimeter

*p = another value ✕	*p = 0; ✕	*p = <value> ✕
*p = a ✕	*p = value; ✕	*p ✕
*p = ✕	int *p = &a ✕	

The correct answer is: *p = <val>;

10

When calling scanf, why do you need to put a & symbol in front of a variable in which you want scanf to store an input value?

Mentimeter

so it assigns the input to the actual variable, not a copy of it

If you want to directly alter a variable rather than a copy of it, you need to provide the pointer to store scanf input

To assign the value to the memory at that variables location

because you need to be able to modify the variable

because it's using a pointer

Because you want to change the value being stored by that variable

To get the adrress of that variable

to refer to the address of the variable

The correct answer is: We pass the address of the variable in scanf, so we can change the variable's value in scanf.

9

However, when we scanf a string: char foo[100]; scanf("%s", foo);, we don't need the & symbol. Why is that?

Mentimeter

because the array is already a pointer object ✗	arrays don’t need it, they always reference the address ✗	c automatically makes foo a pointer to the first element of the char array ✗
because foo is an address to the first element of the string. and %s will access this and read it. ✗	because arrays can be altered within functions ✗	Because it's in the stack ✗
You are already using an array, which acts as a pointer ✗	Because string has already stored adrress ✗	array name already indicate the address ✗

The correct answer is: Because foo is an array of char. foo itself is an address to the first element of the array.

10

Quiz

```c
#include <stdio.h>

int func(float ra[], float x, float *y) {
	ra[0] += 10;
	x *= 20;
	*y += 30;
	return 40;
}

int main() {
	float a = 1;
	float b = 2;
	float c[] = {3, 4, 5, 6};
	int d = func(c, a, &b);
	printf("%.2f, %.2f, %.2f, %d\n", a, b, c[0], d);
}
```

Mentimeter

Trace the program shown on the slide:

1,32,10,40 ✕

1, 32, 13, 40 ✕

1 13 ✕

The correct answer is: 1.00, 32.00, 13.00, 40

4

Ask me anything

Mentimeter

2 questions
1 upvotes

Q & A

- I've seen `**ptr` before. What does this mean?

Answer: It's a pointer-to-pointer. If you want to change the value of a pointer in a function, you will need to pass this pointer by another pointer. In this case, you will use a pointer-to-pointer. e.g.

```c
void initArray(int **a, int asize) {
    if (*a) free(*a);
    *a = malloc(sizeof(int) * asize);
}
int main() {
    int *array = NULL;
    printf("Enter the array size you need:");
    int arraysize = 0;
    scanf("%d", &arraysize);
    initArray(&array, arraysize);
    ...
}
```

We need to pass the address of `array` to `initArray`, so that we can change its value inside the function. We use a pointer-to-pointer for that.

Q & A

- Is there a way to pull from the public repo without typing in our password too! The steps for ssh seem to only for the private one.

Answer: It should work for the public repo as well. Check `.git/config` to see if you have changed it to use git:// instead of https://.

Class exercises

Ex 4-1