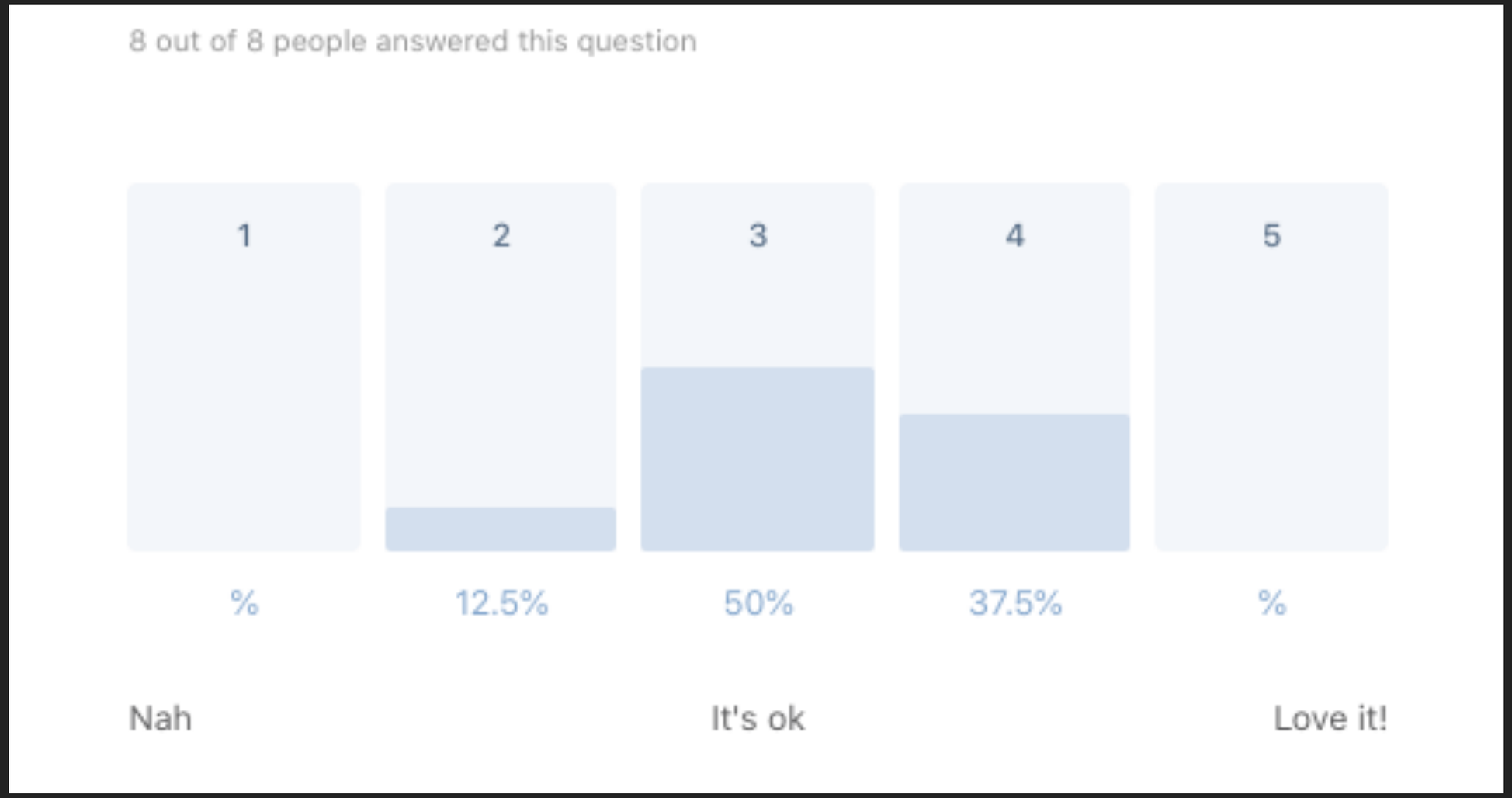
SEPTEMBER 23TH 2020

ELEMENTARY PROGRAMMING

SOME COVID BEST PRACTICES BEFORE WE START

- If you fill ill, go home
- Keep your distance to others
- Wash or sanitise your hands
- Disinfect table and chair
- Respect guidelines and restrictions

FEEDBACK CHECK



NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog4

SOME MORE INFO ON BASIC TYPES – LIMITS

- If you want to know the range for the basic types you can import <limit.h> and use **{TYPE}_MAX** or **{TYPE}_MIN** where {TYPE} is for example int or long

Table 7.2
Integer Types on a 32-bit Machine

Type	*Smallest Value*	*Largest Value*
`short int`	–32,768	32,767
`unsigned short int`	0	65,535
`int`	–2,147,483,648	2,147,483,647
`unsigned int`	0	4,294,967,295
`long int`	–2,147,483,648	2,147,483,647
`unsigned long int`	0	4,294,967,295

In recent years, 64-bit CPUs have become more common. Table 7.3 shows typical ranges for the integer types on a 64-bit machine (especially under UNIX).

Table 7.3
Integer Types on a 64-bit Machine

Type	*Smallest Value*	*Largest Value*
`short int`	–32,768	32,767
`unsigned short int`	0	65,535
`int`	–2,147,483,648	2,147,483,647
`unsigned int`	0	4,294,967,295
`long int`	–9,223,372,036,854,775,808	9,223,372,036,854,775,807
`unsigned long int`	0	18,446,744,073,709,551,615

SOME MORE INFO ON BASIC TYPES – READING AND WRITING INTEGERS

If your variable definition is `unsigned int u`

then your I/O is:

```
scanf("%u", &u);
printf("%u", u);
```

SOME MORE INFO ON BASIC TYPES – READING AND WRITING INTEGERS

If your variable definition is `short s`

then your I/O is:

```
scanf("%hd", &s);
printf("%hd", s);
```

SOME MORE INFO ON BASIC TYPES – READING AND WRITING INTEGERS

If your variable definition is `long l`

then your I/O is:

```
scanf("%ld", &l);
printf("%ld", l);
```

SOME MORE INFO ON BASIC TYPES – READING AND WRITING INTEGERS

If your variable definition is `long long l`

then your I/O is:

```
scanf("%lld", &ll);
printf("%lld", ll);
```

SOME MORE INFO ON BASIC TYPES – READING AND WRITING FLOATING NUMBERS

If your variable definition is `double d`

then your I/O is:

```
scanf("%lf", &d);
printf("%g", d);
```

printf depends on g, e or f

SOME MORE INFO ON BASIC TYPES – READING AND WRITING FLOATING NUMBERS

If your variable definition is `long double d`

then your I/O is:

```
scanf("%Lf", &d);
printf("%Lf", d);
```

printf depends on g, e or f

SOME MORE INFO ON BASIC TYPES – CHAR IS AN INT

```
 1 #include <stdio.h>
 2
 3 int main(void) {
 4   char ch;
 5   printf("tell me one char: ");
 6   scanf("%c", &ch);
 7   if(ch >= 'a' && ch <= 'z') {
 8     ch = ch - 'a' + 'A';
 9   }
10   printf("%c\n", ch);
11   for(char c = 'A'; c <= 'Z'; c++) …
13 }
```

65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w
88	58	130	X	X	120	78	170	x	x
89	59	131	Y	Y	121	79	171	y	y
90	5A	132	Z	Z	122	7A	172	z	z

SOME MORE INFO ON BASIC TYPES – IMPLICIT TYPE CONVERSION

```
long double
    ↑
  double
    ↑
  float
```

```
unsigned long int
        ↑
    long int
        ↑
  unsigned int
        ↑
     int
```

If one of the operand is a floating type then the non floating is converted to floating subtype based on size

SOME MORE INFO ON BASIC TYPES - CASTING

```
(type) expression
```

```
1 #include <stdio.h>
2 int main(void) {
3    float number = 10.5f;
4    float frac_part = number - (int) number;
5    printf("%f\n", frac_part);
6 }
```

SOME MORE INFO ON BASIC TYPES – TYPE DEFINITIONS

```
1  #include <stdio.h>
2  typedef float Kgs;
3  typedef float Lbs;
4  Lbs convert(Kgs weight) {
5    return weight * 2.20462262185f;
6  }
7  int main(void) {
8    Kgs weight = 80.0f;
9    Lbs convertedWeight = convert(weight);
10   printf("%.2f\n", convertedWeight);
11 }
```

This is a new type

SOME MORE INFO ON BASIC TYPES – SIZEOF

- If you want to know how much memory is required to store a value you can use `sizeof(type-name)`
- It represents the number of bytes required as unsigned integer
- You can also do `sizeof(expression)` like `sizeof(i+j)`

ARRAYS

- Arrays are a data structure containing a certain number of data values
- Arrays can have more than one dimension
- A one dimension array could be imagined like this:

ARRAYS

- We can declare an array as `int a[10]`
- Can access elements via indexed like `a[0] = 1`
- You can use them as a normal variable: `a[i]++`

```
1 int main(void) {
2     int n;
3     scanf("%d", &n);
4     int a[n]; //variable length array
5     for (int i = 0; i < n; i++) {
6         a[i] = i;
7     }
8 }
```

ARRAYS

- We initialise an array this way: `int a[10] = {1,2,3,4,5}`
- The elements from 5 to 9 will be equal to 0
- This will initialise everything to the same value `int a[10] = {0}`
- You can define a multidimensional array `int m[2][2] = { {1,2}, {3,4} };`
- You can get the size of an array with sizeof: `sizeof(a)/sizeof(a[0])`

FUNCTIONS

return type*

parameters

variable only valid in function scope

```
1 double average(double a, double b) {
2     int div = 2;
3     return (a+b)/div;
4 }
```

scope

return is used when type is not void

*you cannot return an array

FUNCTION CALL

parameters*

```
average(a, b);
```

*when the function doesn't have any arguments then use empty parenthesis like `function_name();`

FUNCTION DECLARATION

```
1  #include <stdio.h>
2
3  double average(double a, double b);
4
5  int main(void) {
6    double x, y;
7    printf("Tell me two numbers: ");
8    scanf("%lf %lf", &x, &y);
9    printf("The average is %g", average(x, y));
10 }
11
12 double average(double a, double b) {
13   int div = 2;
14   return (a+b)/div;
15 }
```

ARGUMENTS ARE PASSED BY VALUE

n won't be changed outside of this function

```
1  #include <stdio.h>
2  int power(int x, int n) {
3      int result = 1;
4      for (; n > 0; n = n - 1) {
5          result = result * x;
6      }
7      return result;
8  }
9
10 int main(void) {
11     int n = 4;
12     printf("%d\n", power(3, n));
13     printf("%d\n", n); //will print what?
14 }
```

A NOTE ON ARRAYS PASSED AS ARGUMENTS

size of the array

```
1 int sum_array(int a[], int n) {
2     int sum = 0;
3     for (int i = 0; i < n; i++) {
4         a[i] = 4;
5     }
6 }
```

I can always change the value inside the function

RECURSION

```
1 int fact(n) {
2   if (n <= 1) {
3     return 1;
4   } else {
5     return n * fact(n - 1);
6   }
7 }
```

RECURSION

```
fact(3)
|
3 * fact(2)
|
3 * fact(2) * fact(1)
|
3 * fact(2) * 1
|
3 * 2
|
6
```

EXERCISE #1

Summing a Series of Numbers (Revisited)

In Section 6.1, we wrote a program that sums a series of integers entered by the user. One problem with this program is that the sum (or one of the input numbers) might exceed the largest value allowed for an `int` variable. Here's what might happen if the program is run on a machine whose integers are 16 bits long:

```
This program sums a series of integers.
Enter integers (0 to terminate): 10000 20000 30000 0
The sum is: -5536
```

EXERCISE #2

Dealing a Hand of Cards

Our next program illustrates both two-dimensional arrays and constant arrays. The program deals a random hand from a standard deck of playing cards. (In case you haven't had time to play games recently, each card in a standard deck has a *suit*—clubs, diamonds, hearts, or spades—and a *rank*—two, three, four, five, six, seven, eight, nine, ten, jack, queen, king, or ace.) We'll have the user specify how many cards should be in the hand:

```
Enter number of cards in hand: 5
Your hand: 7c 2s 5d as 2h
```

NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog4

SOME COVID BEST PRACTICES BEFORE WE LEAVE

- Disinfect table and chair
- Maintain your distance to others
- Wash or sanitise your hands
- Respect guidelines and restrictions outside