

Day 12 notes

- exercise 11 review
- day 12 recap questions
- exercise 12

Announcements/reminders

- HW2 due *this evening* by 11pm
 - no late submissions!
- HW3 due Friday 2/25 by 11pm

slido.com
jhuintprog01
↑
zero

Exercise 11 review

pairwise_sum.c

When running the program using valgrind:

```
valgrind --leak-check=full ./pairwise_sum
```

a memory leak is reported:

```
==17736== 16 bytes in 1 blocks are definitely lost in loss record 1 of 1
==17736==    at 0x483B7F3: malloc
(in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==17736==    by 0x10922B: pairwise_sum (pairwise_sum.c:28)
==17736==    by 0x109399: main (pairwise_sum.c:57)
```

valgrind indicates there is a memory leak: the memory is allocated in the call to the pairwise_sum function on line 57 of the main function.

Exercise 11 review (continued)

The code:

```
int *pairsum2 = pairwise_sum(pairwise_sum(array, 5), 4);  
...  
free(pairsum2);
```

Issue: pairwise_sum returns a pointer to a dynamically allocated array, but for the "inner" call, the array is never freed.

Fix:

```
int *a = pairwise_sum(array, 5);  
int *pairsum2 = pairwise_sum(a, 4);  
...  
free(pairsum2);  
free(a);
```

Exercise 11 review (continued):

primes.c

Issue: the `set_primes` function needs to call `realloc` if the array of results needs to be increased in size.

However, `realloc` can and usually does return a pointer to a new dynamic array (with a different memory address).

Unless `set_primes` can modify the list pointer in `main`, the `main` function has no way of knowing the address of the re-allocated array.

Trace:

set_primes

main



Exercise 11 review (continued):

Solution: change `set_primes` so that it takes a pointer to the list pointer variable in the main function.

`set_primes`:

```
// originally  
int set_primes( int *list , int capacity )
```

```
// updated  
int set_primes( int **list , int capacity )
```

main:

```
int *list = /* initial allocation of array */
```

```
// originally  
int prime_count = set_primes( list , capacity );
```

```
// updated  
int prime_count = set_primes( &list , capacity );
```

Trace:

set_primes

main



Exercise 11 review (continued):

Changes to set_primes:

Essentially, everywhere that "list" was mentioned, we now want "*list" so that we are referring to the "list" pointer variable in main.

One issue: array subscript operator has higher precedence than the pointer dereference operator (*)

So, instead of changing

```
list[idx++] = n;
```

to

```
*list[idx++] = n;
```

it should be

```
(*list)[idx++] = n;
```

Day 12 recap questions:

1. What output is printed by the “Example code” below?
2. Assume that `arr` is an array of 5 int elements. Is the code
`int *p = arr + 5;`
legal?
3. Assume that `arr` is an array of 5 int elements. Is the code
`int *p = arr + 5; printf("%d\n", *p);`
legal?
4. What output is printed by the “Example code 2” below?
5. Suppose we have variables
`int ra1[10] = { 1, 2, 3}; int * ra2 = ra1;`
and
`int fun(int *ra);`
declarations. Will `fun(ra1);` compile? Will `fun(ra2);` compile?
What if we change the function declaration to
`int fun(const int ra[]);`?

