2016/10/02

1

also void (?)

qsort ((void*)zip, count, sizeof(struct ~~zipType~~ studentArray), compare);

~~student~~

sizeof (student studentArray) ?

or

sizeof (struct student) I think this one.

const void*

(int*)voidA

const int*

int (*voidA) "typecast"

(int*) voidA

compareFunc → use to point @ different starting place in memory

- a qsort(array, 10, Size of(int), compareFunc)

array to be sorted

array size: limit

size of each train car: increments of memory

calling/passing comparison func

2016/10/03 MONDAY

a= [3,3,2,3,1,2,3] how to sort this?

```
{ if a[i1] > a[i2] -----> (a[i] ≷ a[i+1])
      switch ahem;
  elseif a[i1] ≤ a[i2]
      keep the same;
  else
      cout << "bork!\n";
  return statement?
  }
```

```
~~if/ab~~ Counter = 1;
{ if (a[i] > a[i+counter])
      ~~switch them~~; move a[i] after a[i+counter]);
  else
      keep a[i] in the same spot;
  Counter ++;
}
```

int *a; =something → datatype of "something" must be address (another pointer's value)

int* a;

or

*a = something ← DT of "S" ⇒ int (or *a's DT)

```
   0 1 2 3 4 5 6
~~a=[3,3,2,3,2,1,3]~~
  1776 1777 1778 1779 1780 1781 1782
```

```
      0     1     2     3     4     5     6
a = [ 3  ,  3  ,  2  ,  3  ,  2  ,  1  ,  3 ]
    1776  1777  1778  1779  1780  1781  1782
```

```
int ~~counter~~ = 1;
( if )
```

```
void move^Func( int *array, int length) {
    int counter=1;
    if a[i] > a[i+counter]
        ~~a[i] = a[i+counter+1]~~;   //no, that would reassign the value!
  ?     i = i+counter+1;             I want to reassign the index!
```

reroute → try -1, 0, +1 and then see if I can leverege ⓘ into anything functional w/ other func?

```
for (i) {
    if (i == +1)
        do this;
    else if (i == ~~0~~ -1)
        do this;
    else if (i == 0 )
        do nothing;
    else
        cout << "bork /n";
}
```

- ultimately I continue to be unsure on how to rearrange elements of an array. Perhaps reassigning the pointer to a new address? But how is ⓘ actually done?
- Also L? bubble sort for refrence.
  → perhaps have i compare itself down the line? Keep floating towards the right until it's the largest.

```
if i₁ > i₂
    move right, compare to i₃, i₄ . . . iₙ
if i₁ ≤ i₂
    stay in place
```

```
int counter = 1;
if (a[i] > a[counter])
    move right;  (or swap)
    counter ++;  //keep going ~~till~~ while counter < A_SIZE
else if (a[i] ≤ a[counter])
    stay in place; break;  //then move onto next "i".
```

```
p1 = a[0];
p2 = a
```

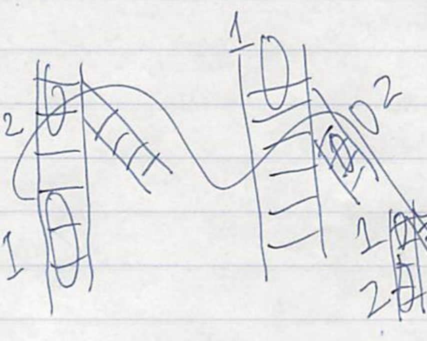
```
a = [2,1];
   if (2 > 1)
pointer→a[0] = a[1];
pointer→a[1] = a[0];
```

```
p0 = &a[0];
p1 = a[1];

if (p0 > p1)  or whatever syntax
              is correct(*, &, etc)
   p0 = p1; (or whatever syntax) } // swap two elements around.
   p1 = p0;
```

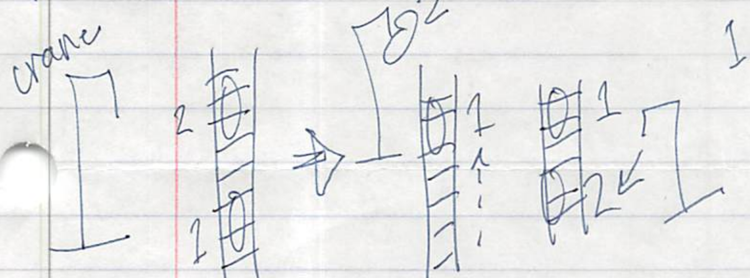
(2016/10/06 Thur)

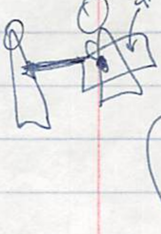
* = value pointed at
& = address of



a[2] = {2,1}

a[0] = a[1];
a[1] = a[0];





```
~~if (a > b)~~ int *a, *b;
   int crane;
   crane = &b;
   *b = &a;
   *a = crane;
```

working w/ ints

AC1 AC2

~~address_card_one, address_card_two~~

```
void swap_func(int* AC1, int* AC2) {
int* a, *b;
int* crane = new int;
cant be on left
&*crane = b;
* crane = &b;
~~a = &b;~~ b
```

		read value of int	write value of int
int*	AC1	x = *AC1	*AC1 = x
int*	AC2	x = *AC2	*AC2 = x

crane

delete crane

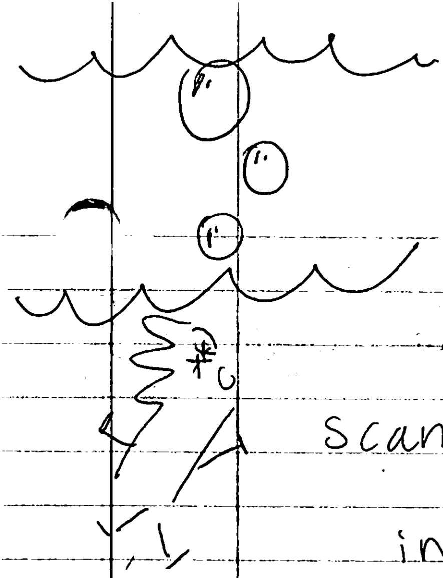
```
int crane;
crane = *b;
*b = *a;
*a = crane;
```

mailbox: needs location in memory!

POINTER { magic tool ⓘ lets us use addresses as if they're values

presence of *

```
______ masterFunc( ) {
   · find the smallest element in array
sortFunc → · if ~~sma~~ a[x] > smallest
             switchFunc( )
   return intArray;
```



25 52 17 91 38 12

compare value of each element against comparVar → set equal to #, ~~but~~ if one's smaller, then make ⊕ new value

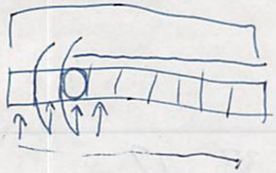
Scan array, look for smallest #, put ⊕ at index Ø
increment indexTally ++
Scan array again, looh for second smallest, put @ indexTally (inder1 in this case)
indexTally ++
and so on.

➤ Should I Sort elements of actual array, of pointers, of array copy? Unknown.

indexTally = Ø;
~~comparVar = Ø;~~

~~@ index~~ compare ~~to~~ a[indexTally =0] to values of each other index in array → if any are smaller, swap them.
indexTally ++;
now Start @ a[indexTally =1] → do the same compare & swap.

L-2



for last x elements, find smallest and place it at a[length-x-1]
repeat for x = x-1

```
void switchFunc(int * pA, int * pB) {
     int crane = *pB;
     *pB = *pA;
     *pA = crane;
}


int * SortFunc(int * intArray, int arrayLength) {
     if (intArray[0] > intArray[1]) {
           SwitchFunc(&intArray[0], &intArray[1]);
     }
     return intArray;
}


int  smallest(int * array) {
     int compVar = 100;
     int indexTrack = 0;
     for (int i = 0; i < 3; i++) {
          if (compVar > array[i]) {
               compVar = array[i];
               indexTrack = i;
          }
     }
     return indexTrack;
}
```

~~aa~~ int a[3] = {3,2,1};

(2016/10/18)

find the smallest!

```
indexTally=0;
compVar=0;
for (int i=~~0~~indexTally; i<aLength; i++) {
        if (compVar < i) {
            compVar = i;
        }
  }
```

(2016/10/20)

```
~~a[0]~~ indexTally = 0;
compVar= 100;  // a bigger number than anything in array
```

could be done w/ more sophistication

- take a[0]. Evaluate a[1] to end → see which element is smallest by comparing them to compVar. Then compare a[0] to ~~compvar~~ a[indexTrack] and swap the two.

    > So, compvar can't be the raw number, but I actually want to track which index is the smallest.

done ✓

~~a[3]; {3, 1, 2};~~

> first, how do I swap two var? Oh duh, I've already done @ part. Man, so stupid!

```
indexTally=1;
for (int i = indexTally; i<aLength; i++) {
   smallest(a);
   }
```

~~Compa~~ set aside a[0]. Evaluate a[1] thru a[end]

↳ find the smallest. Return the index of smallest value.

~~Compare a[0] w/~~

Plug a[0] w/ a[indexTrack] ~~to~~ into ~~swap~~ switch func. which is part of sort func.

→ if a[0] > a[indexTrack] then switch.

thinking about Sagan's raw wonder and delight at the glory of life and learning gets me hot.

OK, I need to find a way to loop through smallest for loop until I'm all the way through the array...

Omaha

outpatient Surgical Specialties Center

25th mon follow-up ~~after 2 wks~~ waiting

MOM

how is this new pen at writing? The ink look and feel are exactly as I would expect → the pen is slightly different. Fun and new and just a little different! :)

2016/10/27

find a way to place beginning of smallest() func at a.length-x-1, and keep shortening @ length until I've iterated thru whole array.

a= 1, 2, 3, 4, 5, 6
0 1 2 3 4 5

hold onto index 0, and run index 1 thru 5 through smallest() func.

→ then I want to compare a_0 to $a_{smallest}$ using sortFunc and maybe switchFunc.

then ++ index by 1 (index 1, now) and run the whole machine again.

```
for a
  do this junk
  ~~item~~ aTrack ++;
  {when (aTrack = ~~a~~ ALENGTH) {
     break;
   }
```

embedded is for loop initial conditions...

```
for(aTrack=0; aTrack < ALENGTH; aTrack++) {
    smallest();
    sortFunc();
    switchFunc();
   aTrack ++;
}
```

2016/10/31 Monday

So I'm increasingly internalizing what my code is doing by default. If I ~~ba~~ try to sort this array:

~~a=[~~ a[4] = {4, 3, 2, 1};

then I end up w/ this result:

[3, 2, 1, 4];

→ the first element compares itself to index 1, then swaps places; then w/ index 2, then swaps; then i3, & swaps. It iterates & compares ~~through~~ & swaps through the whole array.

So, as I extend my sorting/swapping, I need to shorten up from the end of the array, not the beginning. (Instead of indexTally++, it should be endOfArray-- ...or something).

→ → testing this idea...

4321 } expected outcome → 3214 }} but 3241 $\xrightarrow{\text{outcome}}$ 2314
3214 } → 2134 ↳ unexpected
(would have expected 2341 outcome)

I should make an attempt to more fully understand what's happening here → maybe @ will lend itself to me discovering next steps?

>> current code isn't even using smallest() func <<

unexpected!
a=33, 91, 2, 65 → 33, 2, 65, 91
3, 9, 0, 6 → 3, 0, 6, 9 (same result as above)

@ 1324

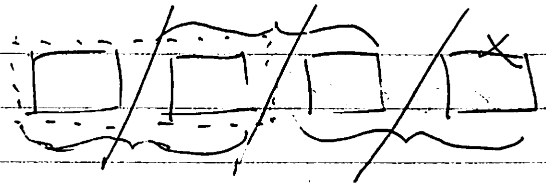
✓ 4321 → 3, 2, 1, 4 i0 moves progressively to end.
✓ 3214 → 2, 1, 3, 4 i0 moves progressively to 2nd-to-final place, then stays put
unexpected 2143 → 1, 2, 3, 4 i0 & i1 swap; i2 & i3 swap
~ fuzzy 1432 → 1, 3, 2, 4 i0 stays in place; i2 moves progressively to the end of array.

✓ 1234 → 1234 all indices stay in place
I think ✓ 2341 → 2314 i0 & i1 stay in place (in order already); i2 & i3 swap
unexpected 3412 → 3124 i0 stays put, i1 moves progressively to end
✓ 4123 → 1234 i0 moves progressively to end

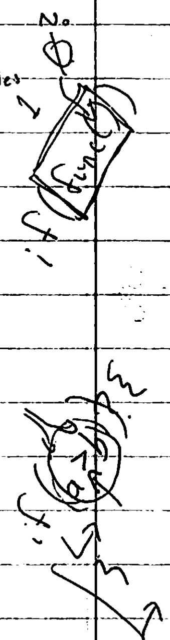
4 (ie, highest number) seems to be doing something special, other #'s sometimes swap, sometimes don't.

if 4 is @ start or end, array is sorted more as expected, but if 4 is in middle, behavior is more muddled?

2016/11/01



a[0]



> take a[0] & a[1], compare the two, then swap them if needed.
> take a[1] & a[2], ~~com~~ ~~w~~ rinse and repeat.
> take a[2] & a[3], ditto
> take a[3] & a[4], ditto

↳ ~~with~~ with this setup, some smaller #'s may get sorted correctly, but once 4 gets caught in the sorting machine, it stays there; the largest # in the array eventually stays stuck in the sorting machinery. This means, long story short, ⓣ in this setup, 4 always gets sorted to the end of the array. Now I've got to find a way to rinse and repeat this whole mess so ⓣ all the numbers get sorted appropriately.

BUBBLE SORT ✓ VERY BASIC

LINE ~~19~~ 20 → turn into separate func → so ⓣ I can change logic depending on the nature of what's being sorted.

✓ LINE 17 → create dummy var to tally backwards as the array is looped through, and ~~then~~ use it to replace arrayLength-1.

2016/11/02

□ Play w/ sorting other types of values: text, multi-dimensional array, etc → not just bald numbers.

□ Make it so ⓣ one of the arguments ~~#~~ sortFunc() takes is ~~an~~ compLogicFunc → so ⓣ I can change ⓣ func argument depending on what type of sort we need.

~~8.5" → just under width of fujitsu~~

21.9cm / 8 5/8" = exact width of fujitsu → for yearbook slicing :)

□ pass func as argument → research.

in declaration (*name) → means this is func pointer

parentheses matter → different from typecast b/c typecast affects variables, not functions.

□ Read more about different sorting algorithms

□ Try to create the sorting algorithm I originally envisioned

2016/11/03

~~void~~ pointer function prototype

```
void func ( void (* f)(int) );
```

2016/11/04

SOURCES ON ALT-RIGHT/CONSERVATIVE (SMART)
LESS WRONG
GEB

2016/11/10

where do I place the pointer and where do I place the actual func?

> It seems like pointer should go to higher-level func arg, and actual func should go to Main.

But added confusion b/c of all the const void* and type casting. Added confusion! I suppose I could simplify the whole process w/ more literal ~~funcs~~, but since I don't fully understand the abstract examples it seems to make it difficult to translate to literals.

And qsort doesn't seem to use pointer func in same way @ Wiki ex does. So how does @ work?! ~~does~~ qsort ~~f~~ have internal hidden magic?

```
a = [1,2,3,4]
~~int add (int* a~~
int add (a, ALENGTH*) {
    for (int i = 0, i < ALENGTH, i++) {
      ~~return a[i] + a[i+1]~~;
    } int sum=0;
}}    sum += a[i];
     }
   return Sum;
}
```

✓ Next step → have secondary func depend on pFunc, then call secondary func in main? ?

in no particular order { Next ~~x~~ next step: turn pointer func into void pointer; translate what I've done to Sandbox 08; make funcs more closely match qsort

PRINT Arthur ØFuzz

2016/11/15

deconstructing qsort (?) can it be done?

(<cstdlib>)

abstract template

```
void qsort (void * base, size_t num, size_t size,
      int (* compar)(const void *, const void *));
```

example→

```
void qsort ( a, ALENGTH, sizeof(int), compare);

int compar (const void * p1, const void * p2);
```

~~example~~ template func

```
int compareMyType ( const void *a, const void *b) {
    if ( *(MyType*) a < *(MyType*) b ) {
        return -1;
    } if ( *(MyType*) a == *(MyType*) b ) {
        return Ø;
    } if ( *(MyType*) a > *(MyType*) b ) {
        return 1;
    }
}
```

example

```
int compareInt (const void * a, const void * b){
    return ( *(int*) a - *(int*)b );
}
```

-OR-

example, closer to my own code.

```
int compareInt(const void * a, const void *b) {
    if (* (int*) a > *(int*) b ) {
        return 1;
    } else {
        return Ø;
    }
}
```

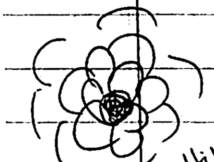
> What are differences btwn qsort code & what I wrote?
the qsort ~~[illegible]~~ pointer func kind of points to itself => there isn't a middle step; rather, the func itself is all void, which allows it to be generic enough to not need a generic middle ground (maybe??). A working theory...

```
int add (const void * array, const void * length) {
    int sum = 0;
    ~~for (*(int *);~~
    for ( int i=0; i < *(int *) length; i++) {
        sum += *(int *) array [i];
    }
    return sum;
}
```

```
int main() {
```

~~next~~ next steps → think more through connections btwn qsort style & what I've got now. What are differences? Can I translate btwn different sets of code? Jim seems to think so.

- Micro beginning step:
  create self-contained printing func in Sandbox11.cpp => instead of ~~[illegible]~~ creating all G clutter in main().



Remember, Hillary didn't win it all in one day: micro steps added together over time equals big wins!

2016/11/16 how do they compare??
- why doesn't compareInt need any args passed?
  > args are void const * → undefined, unchanging (or something...) doesn't matter

· it perhaps depends on some of the other was passed via qsort, along w/ internal magics of qsort internals to work properly.

> but @ seems a little silly, since the sorting logic is supposed to be more self-contained..

: I never have to create secondary func @ defines the args of the pointer func.

confuzed fuzzy > How can I reorient my focus? Break down the problem?

remember: internal variables

this one is super generic

```
int* sortFunc (int * intArray, int aLength, compare) {
    :
```

I guess, how does this internal "compare" var stay generic, yet also apply/adapt to updated/interchangable "comparePointer" in main(); ? Perhaps "compare" should simply point to another func → which is dependent on what's passed in main( )...?

sandbox08.cpp version: "playing around w/ comments..."

Also, I feel like sortFunc in sandbox08.cpp is a mess!

```
:
: for (int h=Ø; h< aLength-1 ; h++) {
        CompareLogic (
```

Go through my array. take first element and compare it to its neighbor. If first element is larger than second, swap them; otherwise keep them in the same spot. Then take the second element in the array, compare it to its neighbor (3rd element, or index 2), and ~~it~~ rinse & repeat.

~~do the~~ the ~~to~~ biggest #@ end; cut off end from 2nd pass, then go at it again

```
int main( ) {

    SortFunc (array, ARRAYLENGTH, ComparePointer);
}
```

→ this one can be intermatched/exchanged w/ other complogic as needed

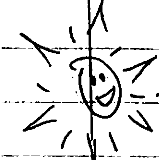
remember: actual variables

>> Future Action Steps:

> continue to demistify qsort → any way to mashup w/ / modify into bubbleNest ??

micro-assignment:?

Create array of letters & print it :)



Inspiration: Bernie's Revolution started w/ obscurity and had zero funding, but it struck a cord w/ American young people and seems to be going places. Bernie is far from the most intelligent, charismatic, Skilled, but he's definitely a hard worker and has passion. Micro Steps; should I be so lucky!

====

pointer function passes the same arguments nested in other functions ...

>> but my comp logic function depends on elements of the array. How can I make it more generic, or otherwise a better multitool w/ my more ~~generic~~ universal interfaces w/ other functions?

LINE 24, 25 ; 43.

how does qsort / compPointer know what to do? Black Boxes!

2016/11/17

```
switch func()                              // acting switch
  crane, varA, varB;

compareFunc()                              // switch logic → Δ depend on nature
    if A > B                               // of what's being sorted.
        1;
    else
        0;
                                           // nitty-gritty insides of sortFund
bubbleNest()
    iterate through ~~arrayLength-1~~ ~~arrayLen~~ shorterVar (ever-shortening array. length var)
        if (point to compareFunc)
            call ~~do~~ switchFunc
    ~~short~~ shorterVar --;

sortFunc()                                 // outer shell of active sort function
    iterate through arrayLength-1
        call ~~perform~~ bubbleNest inside this iteration
    return sorted array;

int main() {                               // main!
    define ARRAY LENGTH;
    define array
    call sortFunc
    print the results
    return Ø;
}
```

[8, 6, 4, 2, 1, 3, 5, 7, 9]

logicFunc

is even?
sort desc.
is odd?
sort ascend.