CS107 Lecture 6

Int-[redacted]. ImplementationNotes

stack.c

```
void StackNew (stack *s)
{
    s->loglength = 0;
    s->alloclength = 4;
    s->elems = malloc (4 * sizeof (int));
    assert (s->elems != NULL);
}
```

malloc

free

```
void StackDispose (stack *s)
{
    free (s->elems);
}
```

free: space identified by address donated back to the heap. free(s) would be wrong because we assume that space for [redacted]s was set aside and we are only freeing up the array that it points to

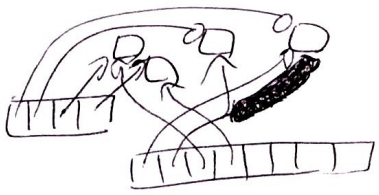
↖ does not work generically, e.g. array of strings

```
void StackPush (stack *s, int value)
{
    if (s->loglength == s->alloclength) {
        s->alloclength *= 2;
        s->elems = realloc (s->elems,
                            s->alloclength *
        assert (s->elems != NULL);          sizeof (int));
    }
    s->elems [s->loglength] = value;
    s->loglength++;
}
```

← works generically

realloc:
- no equivalent in C++
- resize the chunk of memory
- first tries to resize in place
- otherwise calls malloc elsewhere and replicate bit pattern and frees old address then returns address of newly allocated space

realloc on pointers array:



realloc (NULL, ...) same as malloc

malloc & realloc : search the heap to allocate a figure

→ O(size of heap)

→ can be expensive

```
int StackPop (stack *s)
{
   assert (s->logLength >0);
   s->logLength--;
   return s->elems[s->logLength];
}
```

shrink/resize allocation?

realloc ignores the request to shrink allocated space

Generic Implementation

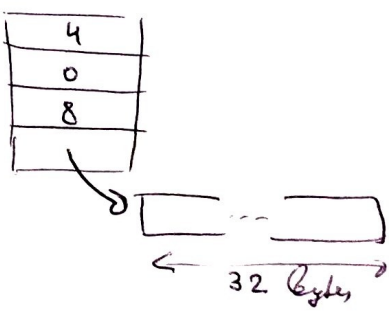
stack.h

```
typedef struct {
   void *elems;
   int elemSize;
   int logLength;
   int allocLength;
} stack;
void StackNew (stack *s, int elemSize);
void StackDispose (stack *s);
void StackPush (stack *s, void *elemAddr);
void StackPop (stack *s, void *elemAddr);
```

stack.c

```
void StackNew (stack *s, int elemSize)
   assert (s->elemSize >0);
{  s->elemSize = elemSize;
   s->logLength = 0;
   s->allocLength = 4;
   s->elems = malloc (4 * elemSize);
   assert (s->elems != NULL);
}
```

e.g. double



```
void StackDispose (stack *s)
{
    .
    .
    .
    free(s->elems);
}
```

← for now assume int, double, char... in the previously allocated figure

```
static void StackGrow (stack *s)
{
    s->allocLength *= 2;
    s->elems = realloc(s->elems, s->allocLength * s->elemSize);
}
```

← private function that should not be advertised outside this file (won't be called from outside)

```
void StackPush (stack *s, void *elemAddr)
{
    if (s->logLength == s->allocLength) {
        StackGrow(s);
    }
    void *target = (char*)s->elems + s->logLength *
                                     s->elemSize;
    memcpy(target, elemAddr, s->elemSize);
    s->logLength++
}
```

why not void* const elemAddr? malloc call without do not make the user be responsible for freeing what is allocated within this fxn

```
void StackPop (stack *s, void *elemAddr)
{
    void *source = (char*)s->elems + (s->logLength - 1) *
                                     s->elemSize;
    memcpy(elemAddr, source, s->elemSize);
    s->logLength--;
}
```

memory allocation consideration:

- any function that allocates, should free it, if possible
- a user of a function should not be responsible for freeing what is allocated within the function
  ↳ reason for passing void * elemAddr in StackPop.

ex: int top;
stack s;
StackNew (&s, sizeof(int));

StackPop (&s, &top);