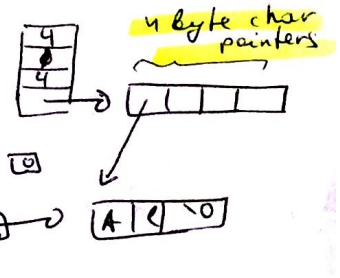
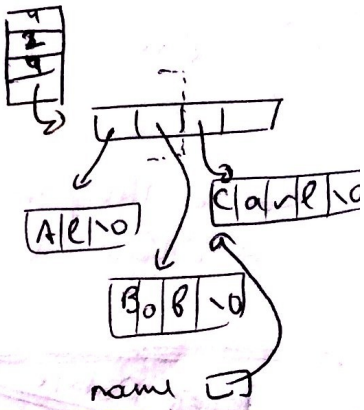
Lecture 7 CS107

ex stack of pointers to strings

```
int main(—, —)
{ const char *friends[] = {"Al", "Bob", "Carl"};
  Stack stringStack;
  StackNew(&stringStack, sizeof(char *));
  for (int i=0; i<3; i++) {
     char *copy = strdup(friends[i]);
     StackPush(&stringStack, &copy);
  }
  char *name;
  for (int i=0; i<3; i++) {
     StackPop(&stringStack, &name);
     printf("%s\n", name);
     free(name);
  }
  StackDispose(&stringStack)
```





Suppose the preceeding for loop was commented out ...

Upgraded version

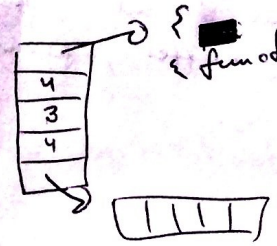
```
void StackNew(stack *s, int elemSize, void (*freefn)(void*));
void StackDispose(stack *s);
void StackPush(stack *s, void *elemAddr)
void StackPop(stack *s, void *elemAddr)
```

information about how to destroy elements when StackDispose is called
NULL if a base type.

in stack.h

```
typedef struct {
   void *elems;
   int elemSize;
   int logLength;
   int allocLength;
   void (*freefn)(void *);
} stack;
```

Store freeing function in struct



```
void StackDispose (stack *s)
{
    if (s->freefn != NULL) {
        for (int i = 0; i < s->logLength; i++) {
            s->freefn ((char *) s->elems + i * s->elemSize);
        }
    }
    free (s->elems);
}
void StringFree (void *elem)
{
    free (*(char **) elem);
}

Stack stringStack
StackNew (&stringStack, sizeof (char *), StringFree);
```

NULL if base types are in the stack array

pointer to a function in the code evaluates function

strings case: dereference and free

Rotate

```
void rotate (void *front, void *middle, void *end)
{
    int frontSize = (char *) middle - (char *) front;
    int backSize = (char *) end - (char *) middle;
    char buffer[frontSize];
    memcpy (buffer, front, frontSize);
    memmove (front, middle, backSize)
    memcpy ((char *) end - frontSize,
            buffer, frontSize);
}
```

same trick to do arithmetic in terms of bytes

buffer with exact # of bytes

cannot use memcpy because overlapping and memcpy does not account for it

call memmove only of have to because the source and destination regions could overlap

Lecture 7 CS107

Quicksort

```
void qsort (void *base, int size, int elemSize,
            int (*cmpfn)(void*, void*));
```

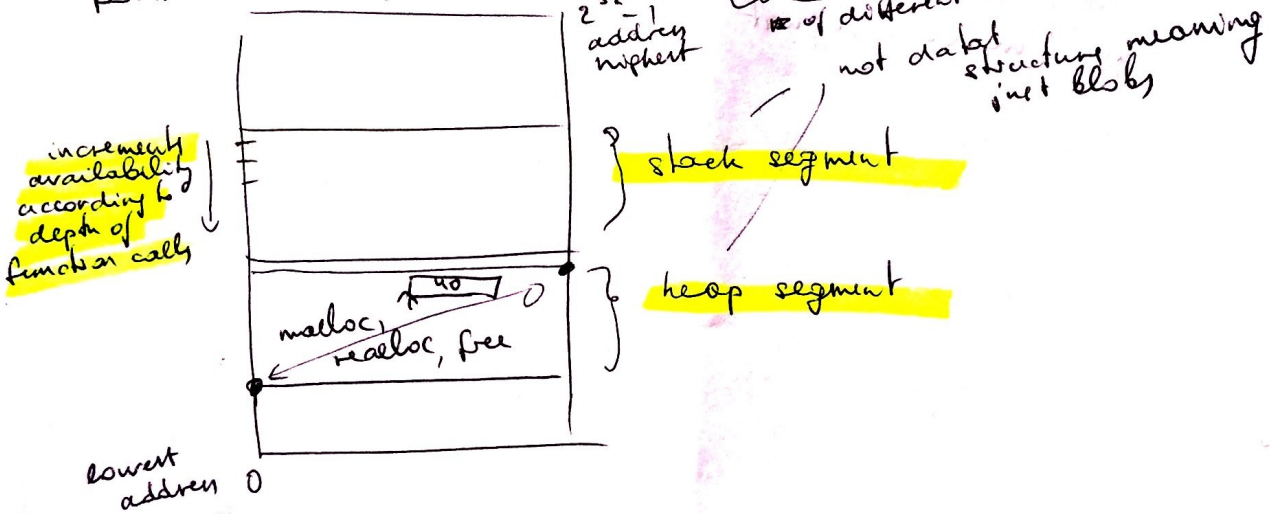
=

> man qsort
> man bsearch

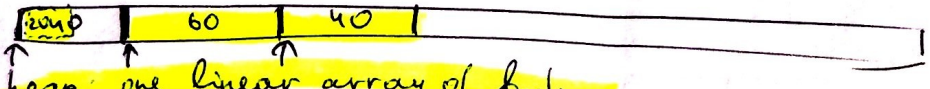
memcpy
memmove
malloc
realloc
free

Implementation of malloc, realloc and free

pointers are 4 bytes, from 0 to $2^{32}-1$ # of different addresses



Focus on heap:



heap: one linear array of bytes

```
void *a = malloc(40);
void *b = malloc(60);
free(a);
```

40 bytes

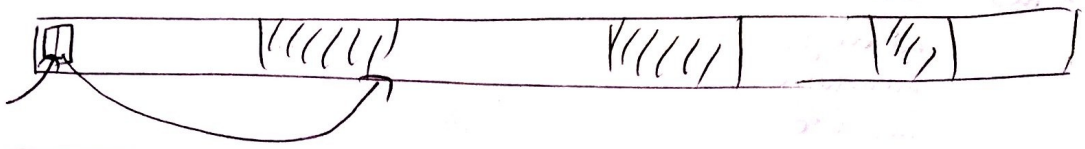
1) starts at the beginning of the heap and searches for an array segment of requested size
2) records that the segment is in use
3) records that the *a blob is available, leaves bit pattern

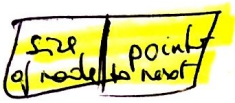
```
void *c = malloc(44);
void *d = malloc(20);
```

well look at the first 40 block but the proceed

Data Structure used by heap manager

link list of free nodes





traverse the linked list to determine which node can accomodate a malloc request