

```
#include <stdio.h> // contains the printf function
#include <stdlib.h> // contains the calloc function
int main(int argc, char** args) {
    // Create an array of 5 integers on the heap, initialized to 0
    int* nums5 = calloc(5, sizeof(int)); Pointer as array

    // access and update with [] works mostly as we expect
    nums5[0] = 100;
    printf("%d\n", nums5[0]);

    nums5[1] = 200;
    printf("%d\n", nums5[1]);

    // except....
    nums5[7] = 600;
    printf("%d\n", nums5[7]);

    int i = 0;
    for(i = -3; i < 8; i += 1) {
        printf("nums5[%d] = %d\n", i, nums5[i]);
    }
}
```

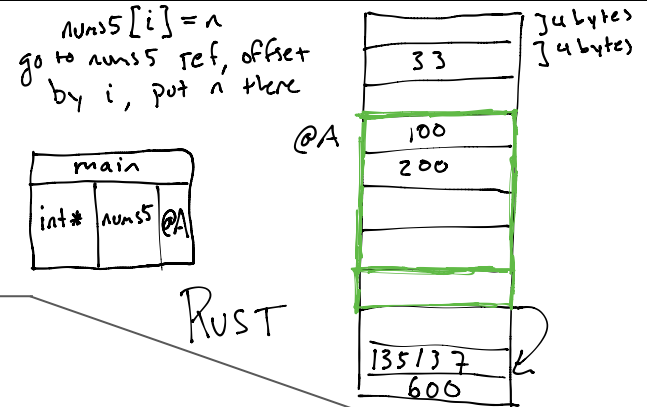
**Danger! Out of bounds indexing**

### calloc(count, size)

Takes a count of a number of elements to allocate space for, and a size representing how much space is needed for each (usually sizeof(Type)), and allocates that many slots on the heap for that data, initialized to zeroes.

### printf(format, value, ...)

Prints the format string with values plugged in for format specifiers. %d is for **decimal** number printing. We will see others.



```
#include <stdio.h>
#include <stdlib.h>
```

// (Strings are coming Friday)

```
typedef struct Entry {
    int key;
    int value;
} Entry;
```

```
int main(int argc, char** args) {
    // Create a single Entry on the heap
    Entry* e = calloc(1, sizeof(Entry));
    // malloc(sizeof(Entry)) would be similar
```

```
// We can use -> to get and update the
// fields of a struct that is allocated on the heap
printf("%d %d\n", e->key, e->value);
e->key = 22;
printf("%d %d\n", e->key, e->value);
e->value = 45;
printf("%d %d\n", e->key, e->value);
```

// The type Entry\* corresponds to a class type in Java:  
// In Java: a reference to an object on the heap  
// In C: a pointer to a struct on the heap

```
Entry** e = calloc(3, sizeof(Entry*));
e[0] = calloc(1, sizeof(Entry));
e[1] = calloc(1, sizeof(Entry));
e[2] = calloc(1, sizeof(Entry));
```

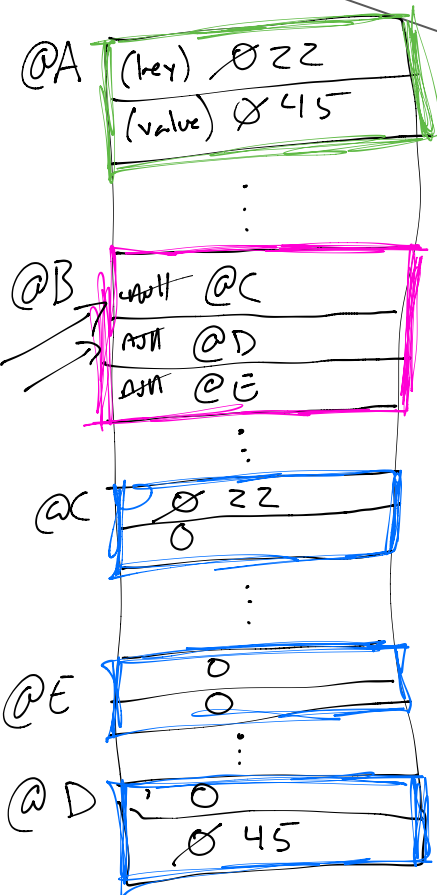
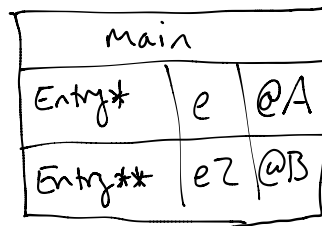
```
printf("%d %d; %d %d\n", e[0]->key, e[0]->value, e[1]->key, e[1]->value);
e[0]->key = 22;
printf("%d %d; %d %d\n", e[0]->key, e[0]->value, e[1]->key, e[1]->value);
e[1]->value = 45;
printf("%d %d; %d %d\n", e[0]->key, e[0]->value, e[1]->key, e[1]->value);
```

// The type Entry\*\* is an array of Entry\*,  
// similar to the type Entry[] in Java.

// We, as programmers, have to remember the different meaning of the two  
// stars based on what \*we\* want.

```
}
```

**Weird! int\* is an array, Entry\* is like a reference, Entry\*\* is an array of references.**



```
#include <stdio.h>
#include <stdlib.h>

typedef struct AList {
    int* contents;
    int size;
    int capacity;
} AList;

AList* make_alist(int start_capacity) {

}

void expandCapacity(AList* alist) {

}

void add(AList* alist, int element) {

}

int get(AList* alist, int index) {

}

void print_alist(AList* alist) {
    int i = 0;
    for(i = 0; i < alist->size; i += 1) {
        printf("%d, ", alist->contents[i]);
    }
}

int main(int argc, char** args) {
    AList* a = make_alist(4);
    add(a, 5);
    add(a, 3);
    add(a, 1);
    printf("%d\n", get(a, 0));
    printf("%d\n", get(a, 1));
    printf("%d\n", get(a, 2));

    print_alist(a);

}
```

```

AList* make_alist(int start_capacity) {
    AList* alist = calloc(start_capacity, sizeof(AList));
    int* contents = calloc(1, sizeof(int));
    alist->contents = contents;
    alist->size = 0;
    alist->capacity = start_capacity;
    return alist;
}

```

```

AList* make_alist(int start_capacity) {
    AList* alist = calloc(1, sizeof(AList*));
    int* contents = calloc(start_capacity, sizeof(int*));
    alist->contents = contents;
    alist->size = 0;
    alist->capacity = start_capacity;
    return alist;
}

```

```

AList* make_alist(int start_capacity) {
    AList** alist = calloc(1, sizeof(AList*));
    int* contents = calloc(start_capacity, sizeof(int));
    alist->contents = contents;
    alist->size = 0;
    alist->capacity = start_capacity;
    return alist;
}

```

```

AList* make_alist(int start_capacity) {
    AList* alist = calloc(1, sizeof(AList));
    int* contents = calloc(start_capacity, sizeof(int));
    alist->contents = contents;
    alist->size = 0;
    alist->capacity = start_capacity;
    return alist;
}

```

```
void add(AList* alist, int element) {
    if(alist.size >= alist.capacity) { expandCapacity(alist); }
    alist[alist->size] = element;
    alist->size += 1;
}
```

```
void add(AList* alist, int element) {
    if(this.size >= this.capacity) { expandCapacity(); }
    alist[alist->size] = element;
    alist->size += 1;
}
```

```
void add(AList* alist, int element) {
    if(alist->size >= alist->capacity) { expandCapacity(alist); }
    alist->contents[alist->size] = element;
    alist->size += 1;
}
```

```
void add(AList* alist, int element) {
    if(alist->size >= alist->capacity) { expandCapacity(alist); }
    alist->contents[alist->size] = element;
    alist->size += 1;
}
```

# Announcements

- Review sessions next week around discussion, in the usual discussion rooms
  - Tuesday 8-**10pm**
  - Friday **4**-6pm
- Resubmit PA7 due Friday of Week 10
- PA8 out today, due next Thursday
  - Part I – go back and measure/improve past PA
  - Part II – implement heaps in C
  - No resubmission for pa8
- Final times
  - 8am Mon/3pm Wed (rooms TBA)
- CAPEs feedback is open
- Last review quiz is special/long, covers week 9 & 10 engagement, will include custom feedback form