Lawrence's Angrave - CS241-02 See-C-Crash

Today: Macros | Pointer arithmetic | printf buffer | malloc/free

Q0. Fix my malloc implementation

could overflow buffer

```
size_t total= 0;
char space[10000];
void* malloc(size_t request) {
        if(request + total > sizeof(space)) return NULL;
        total += request;
        return space + total - request
}
void free(void*ptr) { ? nothing done.
```

Q1. How/why does this macro work?
#define numelements(A) sizeof(A) / sizeof(A[0]) macro → simply text substitution

```
int data [] = {10,20,30};
for(int i = 0; i < numelements(data); i++ ) {... }
```

Q2. Why is this code broken?

```
#define max(a,b)(a)>(b)?(a):(b)
int result = max(10,5) + 1;
printf("Result:%d", result);
```

Q3 Spot the error(s)

```
int f1(int n) {
 int i;= 0 ;
 double* data = malloc( n* sizeof (double*) );   (double)
 while( i < n) data[i++] = 12.3;
} free(data);
```

Q4 Is the following line valid?
printf("%p %p", main, malloc); yes

Q5 What do you think of my logging function? How would you fix it?

```
char buffer[10000];  // GLOBAL
void log(const char* why, const char* mesg) {
 strcat(buffer, why);
 strcat(buffer, mesg); // append to my internal buffer

 if( strlen(buffer) < 500) return; // Do no writing for now
 write(2, buffer, strlen(buffer)); // Send all of the buffer to stderr
 *buffer = 0; // ?
}
```

stderr

Q6 When will printf call write(1, buffer,...) ?

- \n
- exit
- fflush(stdout) / buffer is full

Q7 Pointer arithmetic
Write a function to return the number of items in an int array before a value of -1 is found. Tricky: Use pointer arithmetic (no counters allowed!)

```
count_before(int* array) {
 int* ptr = array;
 while (*ptr != -1) ptr++;
    return (ptr - array) / sizeof(int);
```

Q8 What would you call at line 2 such that p1 can be equal to p2?

```
void* p1 = malloc( 10 );
??____free(p1); p
void *p2 = malloc(8);
```

Solution: after freeing p1 = NULL;

Lawrence's Angrave - CS241-02 See-C-Crash

Today: Macros | Pointer arithmetic | printf buffer | malloc/free

Q0. Fix my malloc implementation

```
size_t total= 0;
char space[10000];
void* malloc(size_t request) {
        if(request + total > sizeof(space)) return NULL;
        total += request;
        return space + total - request
}
void free(void*ptr) { ? knocking store }
```

Q1. How/why does this macro work?

```
#define numelements(A)  sizeof(A) / sizeof(A[0])
```

macro → simply text substitution

```
int data [] = {10,20,30};
for(int i = 0; i < numelements(data); i++ ) {... }
```

Q2. Why is this code broken?

```
#define max(a,b)(a)>(b)?(a):(b)
int result = max(10,5) + 1;
printf("Result:%d", result);
```

Q3 Spot the error(s)

```
int f1(int n) {
 int i;= 0 ;
 double* data = malloc( n* sizeof (double*) );
 while( i < n) data[i++] = 12.3;
} free(data);
```

Q4 Is the following line valid?

```
printf("%p %p", main, malloc);
```

yes

Q5 What do you think of my logging function? How would you fix it?

could overflow buffer

```
char buffer[10000];  // GLOBAL
void log(const char*why, const char*mesg) {
 strcat(buffer, why);
 strcat(buffer, mesg); // append to my internal buffer

 if( strlen(buffer) < 500) return; // Do no writing for now
 write(2, buffer, strlen(buffer)); // Send all of the buffer to stderr
 *buffer = 0; // ?
}
```

stderr

Q6 When will printf call write(1, buffer,...) ?

- \n
- exit
- fflush(stdout) / buffer is full

Q7 Pointer arithmetic

Write a function to return the number of items in an int array before a value of -1 is found. Tricky: Use pointer arithmetic (no counters allowed!)

```
count_before(int* array) {
 int* ptr = array;
 while (*ptr != -1) ptr++;
    return (ptr - array) / sizeof(int);
```

Q8 What would you call at line 2 such that p1 can be equal to p2?

```
void* p1 = malloc( 10 );
??____free(p1); p
void *p2 = malloc(8);
```

Solution: after freeing p1 = NULL;