CSc 352

Malloc, Free, and the Heap

Benjamin Dicken

Announcements

- Spring break
- Hopefully, I won’t see you for a month :)

Recap:

- **`void* malloc(size_t size);`**
  - Allocates **size** bytes and returns the pointer to it, or NULL if failed to alloc
- **`void* calloc(size_t n_items, size_t size);`**
  - Allocates **(n_items* size)** bytes and returns the pointer to it, or NULL if failed to alloc
- **`void free(void * ptr);`**
  - Frees the memory pointer to by **ptr** so that your program can no longer rely on having access to that memory

Activity

Implement the function

- Write a function named **dynamic_strcat**
- Takes two params, **`char*`**s, pointing to two C strings
- Function allocates memory that fits both strings, contacts them, and returns the pointer

More than one value?

- In C, you can return one value from a function (pointer, int, char, etc)

- What if you want to return more than one value?

- For example, a function that:
    - splits a C string exactly in half, and returns both halves
    - Takes a physical address, returns a lat and long value
    - . . . .

Out-Parameters

- An out-parameter is a way of getting a value “out” of a function call without relying on a **return** statement
- If you are calling function Y from function X, you can send Y the address of a local variable from X to store a value into
- This gives the ability to “return” multiple things!

Out-Parameters

- An out-parameter is a way of getting a value “out” of a function call without relying on a **return** statement
- If you are calling function Y from function X, you can send Y the address of a local variable from X to store a value into
- This gives the ability to “return” multiple things!

```c
void split_in_half(char* to_split, char** half_one, char** half_two) {
  int half = (int) (strlen(to_split) / 2);
  *half_one = calloc(1, half+1);
  *half_two = calloc(1, half+1);
  strncpy(*half_one, to_split, half);
  strncpy(*half_two, (to_split+half), half);
}

int main() {
  char alphabet[27] = "abcdefghijklmnopqrstuvwxyz";
  char * h1;
  char * h2;
  split_in_half(alphabet, &h1, &h2);
  printf("alphabet: %s\n", alphabet);
  printf("h1: %s\n", h1);
  printf("h2: %s\n", h2);
  return 0;
}
```

Activity

Implement the function

- Rewrite **dynamic_strcat** to return void, instead give resulting concatenated string back via an out-parameter
- Thus, function should have three total arguments (two “regular” arguments, and one out-param)

Activity

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

#define LARGE 100000

int main() {
  char* longest_line = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest_line == NULL || length > strlen(longest_line)) {
      longest_line = malloc(length);
      strncpy(longest_line, line_buffer, length);
    }
  }
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(line_buffer);
  return 0;
}
```

What does this code accomplish?

Activity

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

#define LARGE 100000

int main() {
  char* longest_line = NULL;
  char* line_buffer = malloc(LARGE);
  while(fgets(line_buffer, LARGE, stdin) != NULL) {
    int length = strlen(line_buffer);
    if (longest_line == NULL || length > strlen(longest_line)) {
      longest_line = malloc(length);
      strncpy(longest_line, line_buffer, length);
    }
  }
  printf("The longest line from standard input is:\n");
  printf("%s\n", longest_line);
  free(line_buffer);
  return 0;
}
```

What is WRONG with how this is written?

Activity

Implement a very simple LinkedList

- In this problem we should implement a very simple linked list
- A node will be represented by:

```
typedef void* lln;
```

- Will have very simple functionality:

```
lln lln_create(int value);
void lln_add(lln node, int value);
void lln_print(lln node);
```

- Then, test it in main!

```c
typedef void* lln;

lln lln_create(int value) {
  ?
}

void lln_add(lln node, int value) {
  ?
}

void lln_print(lln node) {
  ?
}

int main() {
  lln numbers;
  numbers = lln_create(10);
  lln_print(numbers);
  lln_add(numbers, 20);
  lln_add(numbers, 50);
  lln_add(numbers, 30);
  lln_print(numbers);
  return 0;
}
```

```c
typedef void* lln;

lln lln_create(int value) {
  lln node = malloc(sizeof(int) + sizeof(lln));
  if (node == NULL) {
    fprintf(stderr, "ISSUE ALLOCATING NODE\n");
    exit(1);
  }
  int* int_addr = ((int*)node);
  lln* next_addr = (lln)(int_addr+1);
  *int_addr = value;
  *next_addr = NULL;
  return node;
}

void lln_add(lln node, int value) {
  int* int_addr = ((int*)node);
  lln* next_addr = (lln)(int_addr+1);
  if (*next_addr != NULL) {
    lln_add(*next_addr, value);
  } else {
    *next_addr = lln_create(value);
  }
}
```

```c
void lln_print(lln node) {
  int* int_addr = ((int*)node);
  lln* next_addr = (lln)(int_addr+1);
  if (*next_addr != NULL) {
    printf("[NODE (value=%d)] -> ", *int_addr);
    lln_print(*next_addr);
  } else {
    printf("[NODE (value=%d)]\n", *int_addr);
  }
}

int main() {
  lln numbers;
  numbers = lln_create(10);
  lln_print(numbers);
  lln_add(numbers, 20);
  lln_add(numbers, 50);
  lln_add(numbers, 30);
  lln_print(numbers);
  return 0;
}
```