Program memory: dynamic memory allocation

COSC 208, Introduction to Computer Systems, 2021-09-27

Announcements

- Project 1 Part 2 (and revisions to Part 1) due Thursday at 11pm

Outline

- Warm-up
- Pointers as return values
- Program memory
- Heap memory allocation

Warm-up

Q1: *Draw a memory diagram that displays the program's variables and their values just before the* `printf` *statements are executed.*

```
char *split(char *str, char delim) {
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == delim) {
            str[i] = '\0';
            return &str[i+1];
        }
    }
    return NULL;
}

void parse(char *url) {
    char separator = '/';
    char *path = split(url, separator);
    int domainlen = strlen(url);
    int pathlen = strlen(path);
    printf("Domain (%d chars): %s\n", domainlen, url);
    printf("Path (%d chars): %s\n", pathlen, path);
}

int main() {
    char input[] = "colgate.edu/lgbtq";
    parse(input);
}
```

```
Domain (11 chars): colgate.edu
Path (5 chars): lgbtq
```

Pointers as return values

- *What happens?*

  ```
  int *one() {
      int x = 1;
      int *p = &x;
      return p;
  }
  ```

```
int main() {
    int *q = one();
    printf("%d\n", *q);
}
```

  - q points to a variable that no longer exists!
- *So, how can I return a pointer from a function?* — dynamically allocate memory on the heap!

Program memory

- Memory layout

```
┌───────────────────┐ 0
│ Operating system  │
├───────────────────┤
│       Code        │ <- Function instructions
├───────────────────┤
│       Data        │ <- Global variables
├───────────────────┤
│       Heap        │ <- Dyanmically allocated memory
│        ▼          │    Grows toward higher memory addresses
├───────────────────┤
│                   │
│                   │
├───────────────────┤
│        ▲          │    Grows toward lower memory addresses
│       Stack       │ <- Local variables and parameters
└───────────────────┘ Max address
```

  - Stack consists of stack frames --- add a frame when a function is called, remove a frame when a function returns
- Variable storage
  - Local variables and parameters and stored on the stack --- in the frame for the function in which they are declared
  - Global variables are stored in the data section
- Memory allocation
  - Code and data --- automatically allocated with a program starts
  - Stack --- automatically allocated when a function is called; automatically deallocated when a function returns
  - Heap memory --- explicitly allocated and freed by a program

Heap memory allocation

- `void* malloc(unsigned int size)`
- Memory allocated on the heap persists until explicitly freed
- When to `malloc`?
  1. When the amount of space required is not known until runtime
  2. When a value must remain in memory even after returning from a function
- How much to `malloc?
  - Use `sizeof` operator and a type: e.g., `sizeof(int)`
  - How much to malloc for an array? --- multiply `sizeof(type)` by number of elements in array

```
//syntax
type *var = (type*) malloc(size in bytes) //syntax

// allocate a 10-float array
float* arr = (float*) malloc(10*sizeof(float));
  if (arr == NULL) {
    return NULL; // or errcode;
  }
... // do stuff with arr
```

Practice with memory allocation

- Q2: *Write a function called `duplicate` that takes a string (i.e., an array of `char`) as a parameter and returns a copy of that string stored on the heap.*

```c
char *duplicate(char orig[]) {
    char *copy = malloc(sizeof(char) * (strlen(orig) + 1));
    for (int i = 0; i <= strlen(orig); i++) {
        copy[i] = orig[i];
    }
    // Could replace for loop with: strcpy(copy, orig);
    return copy;
}
```

- Q3: *Write a function called `range` that behaves similar to the `range` function in Python. Your function should take an unsigned integer (`length`) as a parameter, and return a dynamically allocated array with `length` unsigned integers. The array should be populated with the values 0 through `length-1`.*

```c
unsigned int *range(unsigned int length) {
    unsigned int *nums = malloc(sizeof(unsigned int) * length);
    for (int i = 0; i < length; i++) {
        nums[i] = i;
    }
    return nums;
}
```

- Q4: *Write a function called `substring` that takes a string, a starting index, and a length, and returns a substring. If the starting index is too large, the function should return `NULL`. If the length is too large, the function should return a shorter substring.*

```c
char *substring(char *str, int start, int length) {
    if (start > strlen(str)) {
        return NULL;
    }
    if (start + length > strlen(str)) {
        length = strlen(str) - start;
    }
    char *substr = malloc(sizeof(char) * (length + 1));
    for (int i = 0; i < length; i++) {
        substr[i] = str[i + start];
    }
    substr[length] = '\0';
    return substr;
}
```

From stack to heap

```c
#include <stdlib.h>

int* copy(int a[], int size) {
   int i, *a2;
   a2 = malloc(size*sizeof(int));
   if (a2 == NULL)
      return NULL;
   for (i = 0; i < size; i++)
      a2[i] = a[i];
   return a2;
}

int main(int argc, char** argv) {
   int nums[4] = {1, 2, 3, 4};
   int* ncopy = copy(nums, 4);
   // .. do stuff with the array ..
   free(ncopy);
   return EXIT_SUCCESS;
```

Extra practice

- Q5: *Write a function called `lengths` that takes an array of strings and the number of elements in the array and returns an array of integers containing the length of each string.*

  ```c
  int *lengths(char *strs[], int count) {
      int *lens = malloc(sizeof(int) * count);
      for (int i = 0; i < count; i++) {
          lens[i] = strlen(strs[i]);
      }
      return lens;
  }
  ```

  - Q6: *Write a funtion called `generate_password` that takes an unsigned integer (`length`) as a parameter, and returns a dynamically allocated array of with `length` randomly selected characters (e.g., uppercase letters, lowercase letters, digits, symbols). Your function should use the `rand()` function from the C standard library, which returns a pseudo-random integer in the range 0 to `RAND_MAX`.*

  ```c
  char *generate_password(unsigned int length) {
      char *password = malloc(sizeof(char) * (length + 1));
      for (int i = 0; i < length; i++) {
          password[i] = (rand() % ('~' - '!')) + '!';
      }
      password[length] = '\0';
      return password;
  }
  ```