Program memory: dynamic memory allocation; malloc

COSC 208, Introduction to Computer Systems, 2022-03-01

Announcements

- Project 1 Part B due Thursday at 11pm

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() {
    //              01234567890123456
    char input[] = "colgate.edu/lgbtq";
    parse(input);
}
```

Pointers as return values

```
int *one() {
    int x = 1;
    int *p = &x;
    return p;
}
int main() {
    int *q = one();
    printf("%d\n", *q);
}
```

There is a problem above... why?

Q2: *Assume you wanted to write a function that creates a copy of a string. What is wrong with each of the following attempts at writing such a function?*

Q2a:

```
char *copy1(char strA[]) {
    char strB[strlen(strA) + 1];
    strcpy(strB, strA);
    return strB;
}
```

Q2b:

```
char copy2(char strA[]) {
    char *strB = malloc(sizeof(char) * (strlen(strA) + 1));
    strcpy(strB, strA);
    return *strB;
}
```

Q2c:

```
char *copy3(char strA[]) {
    char *strB = malloc(sizeof(char *));
    strcpy(strB, strA);
    return strB;
}
```

Practice with memory allocation: `malloc`

Q3: *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.*

Q4: *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`.*

Q5: *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.*

From stack to heap

Q6: *Draw a memory diagram that displays the program's variables and their values.*

```
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;
}
```

free

- `void free(void *block)`
- When to free? — when a value stored on the heap is no longer needed
    - Free memory regions as soon as you are done
    - Do not read/write the memory location after it has been freed!

Q7: *What do the following two functions do? How are they different?*

```
void swap1(int *m, int *n) {
    int tmp = *n;
    *n = *m;
    *m = tmp;
}
void swap2(int **x, int **y) {
    int *tmp = *y;
    *y = *x;
    *x = tmp;
}
```

Extra practice

QA: *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.*

QB: *Write a function 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`.*