Program memory: arrays & pointers; malloc

COSC 208, Introduction to Computer Systems, 2022-02-22

Announcements

- Exam 1 this Thursday

Outline

- Warm-up
- Arrays & pointers
- Pointers as return values
- Program memory
- Heap memory allocation

Warm-up

Q1: *What is the output of this program?*

```
void increment1(int a) {
    a = a + 1;
}

void increment2(int *b) {
    *b = *b + 1;
}

int main() {
    int x = 1;
    int *y = &x;
    increment1(x);
    printf("%d %d\n", x, *y);
    increment2(y);
    printf("%d %d\n", x, *y);
}
```

```
1 1
2 2
```

Stack memory layout

Q2: *What is the output of this program?*

```
int main() {
    int a = 1; // Assume at 0x4
    int *x = &a; // Assume at 0x8
    int **y = &x; // Assume at 0xC
    printf("%p %p %p\n", a, x ,y);
    printf("%p %p\n", *x , *y);
}
```

Output

```
0x1 0x4 0x8
0x1 0x4
```

Arrays & pointers

- An array variable is a pointer to a region of memory where the items in the array are stored

Example

```
int main() {
    char word[] = "hat";
    printf("word = %s\n", word);
    char *ptr = word;
    printf("ptr = %s\n", ptr);
    if (ptr == word) {
        printf("ptr == word\n");
    }
    else {
        printf("ptr != word\n");
    }
    word[1] = 'i';
    printf("word = %s\n", word);
    *ptr = 's';
    printf("word = %s\n", word);
    ptr[1] = 'a';
    printf("word = %s\n", word);
}
```

- This explains why there is no out-of-bounds checks for arrays: the memory addresses in pointers are never checked to see if they are valid

- This also explains why you can change an array within a function and have those changes reflected outside of the function

```
int update(char str[]) {
    str[0] = 'p';
}
int main() {
    char word[] = "hat";
    update(word);
    printf("%s\n", word);
}
```

Q3: *What is the output of this program?*

```
int main() {
    int nums[4] = {1,2,3,4};
    printf("%d %d\n", *nums, nums[1]);
    int *ptr = &nums[1];
    nums[1] += 4;
    printf("%d %d\n", *ptr, nums[0]);
    ptr = (nums + 2);
    printf("%d\n", *ptr);
    ptr++; // num++ is illegal
    printf("%d\n", *ptr);
}
```

```
1 2
6 1
3
4
```

Q4: *What is the output of this program?*

```
int main() {
    char *first = "Colgate";
    char second[10] = "Univ";
    char *f = &first[3];
    printf("%d\n", strlen(f));
    char *s = second;
    *s = 'K';
    s++;
    *(s+2) = 't';
    printf("%s %s\n", second, s);
}
```

```
4
Knit nit
```

Pointers as return values

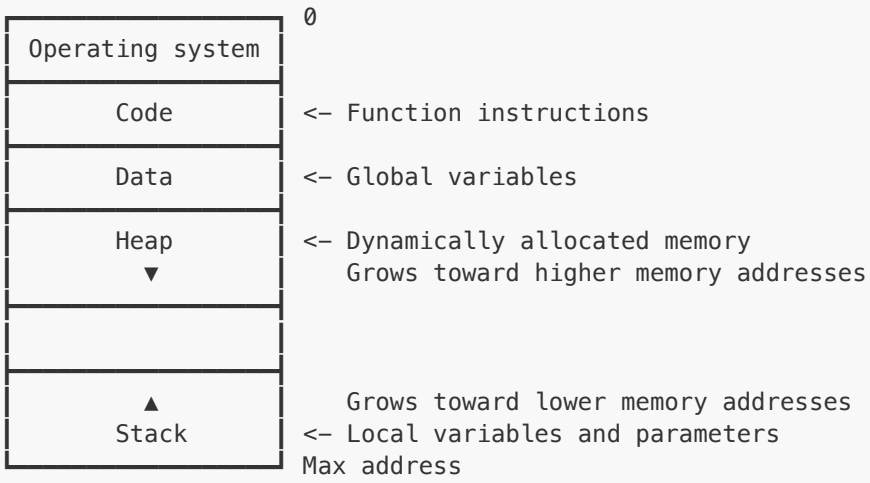
- *What happens?*

```c
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



    - 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

malloc

- `void* malloc(unsigned int size)`
- Memory allocated on the heap persists until explicitly freed
- When to malloc?
    - When the amount of space required is not known until runtime
    - When a value must remain in memory even after returning from a function
- How much to malloc?
    - Use `sizeof` and a type: e.g., `sizeof(int)`
    - How much to malloc for an array? — multiply sizeof(type) by number of elements in array
- Q5: *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.*

```
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;
}
```

- Q6: *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`.*

```
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;
}
```

Q7: *Draw a memory diagram that displays the program's variables and their values when the program reaches the comment `STOP HERE`.*

```
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);
    // STOP HERE
    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);
}
```

Q8: *What do the following two functions do? How are they different?*

```c
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;
}
```

What is the output of this program?

```c
int main() {
    int a = 1;
    int b = 2;
    int *ptrA = &a;
    int *ptrB = &b;
    swap1(ptrA, ptrB);
    printf("%d %d\n", a, b);
    swap2(&ptrA, &ptrB);
    printf("%d %d %d %d\n", a, b, *ptrA, *ptrB);
}
```

```C
2 1
2 1 1 2
```

Extra practice

- QA: *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`.*

```
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;
}
```

- QB: *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.*

```
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;
}
```

- QC: *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.*

```
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;
}
```

Worksheet created by Professor Aaron Gember-Jacobson