Program memory: free; structs

COSC 208, Introduction to Computer Systems, 2021-09-29

Announcements

- Project 1 Part B (and revisions to Part A) due tomorrow at 11pm

Outline

- Warm-up
- Heap memory deallocation
- Pointers to structs

Warm-up

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?

- Q1:

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

  You cannot return a locally-declared array
- Q2:

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

  Returns the first character in the copy, instead of an array of characters
- Q3:

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

  Allocates space for a pointer, not space for the number of characters in strA

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!

- Q4: *What memory deallocation mistake has been made in this code snippet?*

```
int *ptrA = malloc(sizeof(int) * 3);
int *ptrB = ptrA;
free(ptrA);
free(ptrB);
```

  Double free — `ptrA` and `ptrB` point to the same heap region

- Q5: *What memory deallocation mistake has been made in this code snippet?*

```
int *ptr = malloc(sizeof(int) * 3);
ptr[0] = 1;
free(ptr);
ptr[1] = 2;
```

  Access after free — heap region is freed, then used again

- Q6: *What memory deallocation mistake has been made in this code snippet?*

```
int *ptr = malloc(sizeof(int) * 3);
ptr++;
free(ptr);
```

  Not pointing to beginning of allocated region when calling free

- Q7: *What memory deallocation mistake has been made in this code snippet?*

```
int *ptrA = malloc(sizeof(int) * 3);
int *ptrB = ptrA;
ptrA[0] = 0;
ptrB[1] = 1;
free(ptrA);
ptrB[2] = 2;
```

  Access after free — `ptrA` and `ptrB` point to the same heap region

Pointers to structs

- *How do you get a pointer to a struct?*
  - Use address-of (`&`) operator with a parameter/local variable

```
struct tvshow {
    char name[100];
    int season;
};
```

```
int main() {
    struct tvshow favorite = {"This Old House", 42};
    struct tvshow *ptr = &favorite;
}
```

- Allocate space on the heap

```
int main() {
    struct tvshow *ptr = malloc(sizeof(struct tvshow));
}
```

- *How do you access a struct's fields through a pointer to the struct?*
  - `(*ptr).field`

```
printf("There are %d seasons of %s\n", (*ptr).seasons, (*ptr).name);
```

    - Don't do `*ptr.field` — it will try to dereference `field` not `ptr`, because `.` has higher precedence than `*`
  - `ptr->field`

```
printf("There are %d seasons of %s\n", ptr->seasons, ptr->name);
```

- Assume you are given the following code:

```
struct account {
    int number; // Account number
    int balance; // Current account balance
};
int deposit(struct account *acct, int amount);
int transfer(struct account *from, struct amount *to, int amount);
```

  - Q8: *Write the `deposit` function, which adds `amount` to the balance of `acct`. The function should return the amount deposited.*

```
int deposit(struct account *acct, int amount) {
    acct->balance += amount;
    return amount;
}
```

  - Q9: *Write the `transfer` function which moves `amount` from one account to another. The function should return the amount transferred if the transfer was successful or 0 otherwise.*

```
int transfer(struct account *from, struct amount *to, int amount) {
    if (from->balance < amount)
        return 0;
    from->balance -= amount;
    to->balance += amount;
    return amount;
}
```

Extra practice

- *Two structs have been defined representing a queue and an item on a queue.*

```
struct item {
    int value;
    struct item *next;
};
struct queue {
    struct item *head;
    struct item *tail;
};
```

The `new_queue` function creates a new, empty queue.

```
struct queue *new_queue() {
    struct queue *q = malloc(sizeof(struct queue));
    q->head = NULL;
    q->tail = NULL;
    return q;
}
```

- Q10: *Write a function called `enqueue` that adds a new value at the end of the queue.*

```
void enqueue(int value, struct queue *q) {
    struct item *i = malloc(sizeof(struct item));
    i->value = value;
    i->next = NULL;
    if (q->tail == NULL) {
        q->head = i;
        q->tail = i;
    } else {
        q->tail->next = i;
        q->tail = i;
    }
}
```

- Q11: *Write a function called `dequeue` that removes and returns the value at the head of the queue. The function should return -1 if the queue is empty.*

```
int dequeue(struct queue *q) {
    if (NULL == q->head) {
        return -1;
    }
    struct item *i = q->head;
    int v = i->value;
    q->head = i->next;
    if (q->head == NULL) {
        q->tail = NULL;
    }
    free(i);
    return v;
}
```

- Q12: *Write a function called `free_queue` that empties and frees a queue.*

```c
void free_stack(struct queue *q) {
    while (q->head != NULL) {
        dequeue(q);
    }
    free(q);
}
```