CS50 Section Notes

Zad Chin

Table of contents

Preface

This website contains section notes for COMPSCI 50, an Introduction to Computer Science course at Harvard University taught by Professor David Malan. These notes are created by Zad Chin.

Logistics

- Course Website
- Sections: 7:00 - 8:20 pm EDT on Fridays at Zoom
- Office Hours: 7:00 - 8:20 pm EDTon Fridays at Zoom
- Manual Pages : website here
- EdStem Discussion: Ed link

Notes

These section notes will be presented as an online book, and the source for this book at https://zadchin.github.io/CS50Section/. Any typos or errors can be reported at https://github.com/zadchin/CS50Section/issues. Thanks for reading.

This is a Quarto book. To learn more about Quarto books visit https://quarto.org/docs/books.

Section 1

Last Updated: 27 OCT 2023

Date: 27 Oct 2023

This Week

In this section, we will discuss:

- Variables
- Types
- Loops
- Conditions
- Functions

Slides

Slides deck are available here: Slide Section 1

Next Week

Next week, we will discuss:

- Array
- String
- Command Line Argument

Section 2

Last Updated: 10 Nov 2023

Date: 10 Nov 2023

This Week

In this section, we will discuss:

- Array
- String
- Command Line Argument

Slides

Slides deck are available here: Slide Section 2

Section Practice Problems

Section Problem 1

Background

In a classroom, a teacher keeps track of the attendance of students. Each student is assigned a seat number, and their attendance status is recorded as present (1) or absent (0) for a particular day.

Task

Write a program in C that allows the teacher to enter the attendance status for each student and then displays the total number of students present and absent on that day.

Demo

Demo in Section

Starter Code

Copy and following the following code to a new C file in CS50 codespace to start coding!

```
#include <cs50.h>
#include <stdio.h>

#define CLASS_SIZE 5

int main(void) {
    int attendance[CLASS_SIZE];
    int present = 0;
    int absent = 0;

    // Ask the teacher to enter attendance
    printf("Enter the attendance for each student (1 for present, 0 for absent):\n");
    for (int i = 0; i < CLASS_SIZE; i++) {
        // Use get_int() to get input from user
        attendance[i] = get_int("Student %d: ", i + 1);

        // TODO: Add code to count presents and absents
    }

    // TODO: Add code to display the total presents and absents

    return 0;
}
```

Solution

```
#include <cs50.h>
#include <stdio.h>

#define CLASS_SIZE 10

int main(void) {
    int attendance[CLASS_SIZE];
    int present = 0;
    int absent = 0;

    printf("Enter the attendance for each student (1 for present, 0 for absent):\n");
    for (int i = 0; i < CLASS_SIZE; i++) {
        // Use get_int() to get input from user
        attendance[i] = get_int("Student %d: ", i + 1);

        // Tally presents and absents
        if (attendance[i] == 1) {
            present++;
        }
        else if (attendance[i] == 0)  {
            absent++;
        }
        else {
            printf("Error: Attendance needs to be 0 or 1 \n");
            return 1;
        }
    }

    // Display the total presents and absents
    printf("Total present: %d\n", present);
    printf("Total absent: %d\n", absent);

    return 0;
}
```

Section Problem 2

Background

Gen Z often use acronyms to help remember lists or sequences, for example:

- YOLO- You only live once
- FOMO – Fear of missing out
- GOAT – Greatest of all time

Let's create a program that generates an acronym from a list of words.

Task

Write a program in C that:

Asks the user to input a certain number of words, stored in an array of strings. Generates an acronym by taking the first letter of each word and concatenating them. Converts the acronym to uppercase. Prints out the final acronym.

Demo

Demo in Section

Starter Code

Copy and following the following code to a new C file in CS50 codespace to start coding!

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define WORD_COUNT 4 // Number of words in the acronym

int main(void) {
    string words[WORD_COUNT];

    //TODO: Initialize acronym

    // Prompt user for words
    for (int i = 0; i < WORD_COUNT; i++) {
        words[i] = get_string("Enter word %d: ", i + 1);
    }

    // TODO: Generate the acronym from the first letter of each word

    // TODO: Print the acronym

    return 0;
}
```

Solution

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define WORD_COUNT 4 // Number of words in the acronym

int main(void) {
    string words[WORD_COUNT];
    char acronym[WORD_COUNT + 1]; // +1 for the null terminator

    // Prompt user for words
    for (int i = 0; i < WORD_COUNT; i++) {
        words[i] = get_string("Enter word %d: ", i + 1);
    }

    // Generate the acronym from the first letter of each word
    for (int i = 0; i < WORD_COUNT; i++) {
        // Check if the first character is a letter
        if (isalpha(words[i][0])) {
            acronym[i] = toupper(words[i][0]);
        }
        else {
            // Handle the case where the first character is not a letter
            // Placeholder character or print error
            acronym[i] = 'X';
        }
    }

    // Null-terminate the acronym string
    acronym[WORD_COUNT] = '\0';

    // Print the final acronym
    printf("The acronym is: %s\n", acronym);

    return 0;
}
```

Section Problem 3

Background and Task

For the acronym generator, the words that form the acronym can be passed as command-line arguments. Recode the origin code files to take words from command-line arguments to create the acronym.

Demo

Demo in Section

Starter Code

Copy and following the following code to a new C file in CS50 codespace to start coding!

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(int argc, string argv[]) {
    // TODO: Fill up ... to make sure users give at least a certain argc
    if (...) {
        printf("Usage: ./acronym word1 word2 ... wordN\n");
        return 1;
    }

    // TODO: Generate and print the acronym from command-line arguments

    return 0;
}
```

Solution

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(int argc, string argv[]) {
    if (argc < 2) {
        printf("Usage: ./acronym word1 word2 ... wordN\n");
        return 1;
    }

    for (int i = 1; i < argc; i++) {
        if (isalpha(argv[i][0])) {
            printf("%c", toupper(argv[i][0]));
        }
    }
    printf("\n");

    return 0;
}
```

Next Week

Next week, we will discuss:

- Structures in C
- Sorting
- Searching
- Recursion

Note: I am out for the next section.Margaret Tanzosh will replace me for the section. Refer her materials.

Section 4

Last Updated: 8 Dec 2023

Date: 8 Dec 2023

This Week

In this section, we will discuss:

- Pointers
- Memory Allocation (malloc)
- File I/O

Slides

Slides deck are available here: Slide Section 4

Section Practice Problems

Section Problem 1

Background

Write a function in C that swaps the values of two integers using pointers.

Task

- Understanding pointers is crucial in C programming as they allow for efficient manipulation of variables and memory.
- The swap function demonstrates a fundamental use of pointers to directly modify the values of variables at their memory addresses.
- Instead of swapping values using a third temporary variable, this function will directly exchange the values at the memory addresses pointed to by the pointers a and b.

Demo

Demo in Section

Starter Code

Copy and following the following code to a new C file in CS50 codespace to start coding!

```
#include <stdio.h>

void swap(int* a, int* b) {
    // TODO: Implement the logic to swap the values of a and b
}


int main() {
    int x = 10;
    int y = 20;

    printf("Before swap: x = %d, y = %d\n", x, y);
    //TODO: call on the function swap for x and y: swap(...)

    printf("After swap: x = %d, y = %d\n", x, y);

    return 0;
}
```

Solution

```
#include <stdio.h>

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 10;
    int y = 20;

    printf("Before swap: x = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("After swap: x = %d, y = %d\n", x, y);

    return 0;
}
```

Section Problem 2

Background

Implement a "Brightness Adjustment" filter that modifies the brightness of an image. This filter will either increase or decrease the brightness of every pixel in the image based on a given factor.

- Brightness is determined by the intensity of the RGB components of a pixel.
- Increasing brightness involves adding a constant value to each color component (Red, Green, Blue) of a pixel.
- Decreasing brightness involves subtracting a constant value from each color component.
- It's crucial to ensure that the resulting color values are capped between 0 and 255.

Task

Implement a function adjust_brightness in helpers.c to increase or decrease the brightness of the image. The function should take an additional parameter, brightness_factor, to control the adjustment level.

Demo

Will be shown in section

To Test

1. Run `make filter`
2. Run this `filter-practice/ $ ./filter -g images/yard.bmp out.bmp` to see the original file in out.bmp
3. Run this `filter-practice/ $ ./filter -b images/yard.bmp out.bmp` to see the adjusted brightness photo in out.bmp

Starter Code

Download the starter code file here

Pseudocode

```
function adjust_brightness(height, width, image, brightness_factor)
    for each row in image
        for each pixel in row
            for each color component (red, green, blue) in pixel
                new color value = original color value + brightness_factor
                if new color value > 255
                    new color value = 255
                if new color value < 0
                    new color value = 0
                update pixel's color component with new color value
```

Solution

Solution 1:

```
void adjust_brightness(int height, int width, RGBTRIPLE image[height][width], int bright
{
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            image[i][j].rgbtRed = fmin(fmax(image[i][j].rgbtRed + brightness_factor, 0),
            image[i][j].rgbtGreen = fmin(fmax(image[i][j].rgbtGreen + brightness_factor,
            image[i][j].rgbtBlue = fmin(fmax(image[i][j].rgbtBlue + brightness_factor, 0
        }
    }
}
```

Solution 2

```
void adjust_brightness(int height, int width, RGBTRIPLE image[height][width], int bright
{
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            // Adjust Red, ensuring it remains within 0-255 range
            int newRed = image[i][j].rgbtRed + brightness_factor;
            image[i][j].rgbtRed = newRed > 255 ? 255 : (newRed < 0 ? 0 : newRed);

            // Adjust Green, ensuring it remains within 0-255 range
            int newGreen = image[i][j].rgbtGreen + brightness_factor;
            image[i][j].rgbtGreen = newGreen > 255 ? 255 : (newGreen < 0 ? 0 : newGreen)

            // Adjust Blue, ensuring it remains within 0-255 range
            int newBlue = image[i][j].rgbtBlue + brightness_factor;
            image[i][j].rgbtBlue = newBlue > 255 ? 255 : (newBlue < 0 ? 0 : newBlue);
        }
    }
}
```

Homework tips

Homework tips are available in the slide: Slide Section 4

Next Week

For the following section, we will discuss:

- Algorithms!

Section 5

Last Updated: 23 Dec 2023

Date: 23 Dec 2023

This Week

In this section, we will discuss:

- Linked List
- Hash Tables
- Queue, Stack

Slides

Slides deck are available here: Slide Section 5

Section Practice Problems

Section Problem 1

Background

Linked lists are a fundamental data structure in computer science, often used to store sequences of elements. They consist of nodes, where each node contains a piece of data and a pointer to the next node in the list. This structure allows for efficient insertion and deletion of elements. Understanding pointers is crucial for managing linked lists, as they are used to navigate and modify the list structure.

Task

Create a function that reverses a singly linked list. The list is represented by a `struct` named `node`, which contains an integer `number` and a pointer to the next node. Your task is to write a function `void reverse(node **head)` that takes a pointer to the head of the list and reverses

the order of the nodes. The challenge is to manipulate the pointers in each node to reverse the order without creating any new nodes or losing any existing ones.

Demo

Demo in Section

> **Starter Code**
>
> Copy the code at here

Tips & Pseudocode

Example in Words
Imagine a simple linked list: 1 -> 2 -> 3 -> null

1. Start: `prev` = null, `current` = 1, `next` = null
2. First Iteration:
    - next points to 2 (saving link to next node)
    - Reverse link: 1 <- current, but next (2) -> 3
    - Move prev to 1, current to 2 (advance pointers)
    - Now: null <- 1 <- current (2) -> 3
3. Second Iteration:
    - next points to 3
    - Reverse link: 2 <- current, but next (3) -> null
    - Move prev to 2, current to 3
    - Now: null <- 1 <- 2 <- current (3)
4. Final Iteration:
    - next is null (end of list)
    - Reverse link: 3 -> null
    - current becomes null, loop ends
    - Final list: null <- 1 <- 2 <- 3

The head of the list now points to 3, achieving the reversed list: 3 -> 2 -> 1 -> null
Pseudocode

```
1. Initialize three pointers: prev (null), current (head of the list), and next (null).

2. Loop while current is not null:
 - Set next to current->next (save the next node).
 - Redirect current->next to prev (reverse the link).
 - Move prev to current (advance prev up the list).
 - Move current to next (advance current up the list).

3. After the loop, set the head of the list to prev.
```

Solution

```
void reverse(node **head) {
    node *prev = NULL;
    node *current = *head;
    node *next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}
```

Section Problem 2

Background

Hash tables are powerful data structures that allow for efficient data retrieval. They map keys to values, using a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. This exercise focuses on understanding how to search for items in a hash table, an essential operation known as a lookup.

Task

Write a function, `bool lookup(char* word, node* hashtable[], int size)`, that searches for a given word in a hash table. The hash table is represented as an array of linked lists (`node* hashtable[]`), and `size` is the number of buckets in the table. The function should return `true` if the word is found and `false` otherwise.

Demo

Will be shown in section

Starter Code

Copy the code at here

Pseudocode

```
1. Hash the Word using the provided hash function: Use the hash function on the input wo

2. Traverse the Linked List: At the calculated index, traverse the linked list attached
   - If the current node's word matches the input word, return True.
   - If not, move to the next node in the list.

3. Word Not Found: If the end of the list is reached without finding the word, return Fa
```

Solution

Solution 1:

```
bool lookup(const char* word, node* hashtable[]) {
    unsigned int index = hash(word);
    node *cursor = hashtable[index];
    while (cursor != NULL) {
        if (strcmp(cursor->word, word) == 0) {
            return true;
        }
        cursor = cursor->next;
    }
    return false;
}
```

Solution 2

```
bool lookup(const char* word, node* hashtable[]) {
    unsigned int index = hash(word);

    // Use a for loop to traverse the linked list
    for (node *cursor = hashtable[index]; cursor != NULL; cursor = cursor->next) {
        if (strcmp(cursor->word, word) == 0) {
            return true; // Word found
        }
    }
    return false; // Word not found
}
```

Homework tips

Homework tips are available in the slide: Slide Section 5

Next Week

For the following section, we will discuss:

- Python!