COS10007

Week 9 Prac

Question 1

The advantage of breaking down your source files into multiple chunks

(separate .c and .h files) is that you can recompile one chunk at a time and

reuse the chunks in other programs later.

You are also able to use ifdef/ifndef conditions to include different versions

of the file for different operating systems.

Question 2

All of these weeks could have been split into multiple files:

W5Q2

W7Q2

W8Q2

All of these programs had a ton of functions that could exist anywhere and

weren't specific to the task at hand, they all accept arguments instead of

referring to global variables directly.

Question 3

// Week9Question3.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "Week9Question3.h"

int main(void) {

// Take user input.

printf("Please enter a sentence no longer than 100 characters.\n");

char input[100];

fgets(input, 99, stdin);

//printf(input);

printf("\n");

// There is a bug where the program will consider leading whitespace to be a

// word instead of ignoring it and starting at the first valid character but

// I can't be bothered fixing it.

// Cut the words up into individual strings and place them into the stack.

size\_t cache\_pos = 0;

char cache[100] = "";

for(int i = 0; i < strlen(input); i++) {

if(input[i] != ' ' && input[i] != '\0' && input[i] != '\n') {

// Valid character that is part of the word.

cache[cache\_pos] = input[i];

cache\_pos++;

} else {

// End of the word.

printf("%s\n", cache);

// Push the word to the top of the stack.

strcpy((pushWord())->word, cache);

// Reset the cache.

cache\_pos = 0;

for(int a = 0; a < 100; a++) {

cache[a] = '\0';

}

// Skip the rest of the whitespace if it's not a single character.

while(input[i + 1] == ' ') {i++;}

}

}

// Separate the two stacks.

printf("\n");

// Print and free the stack.

sent\_ptr temp;

while(head != NULL) {

temp = popWord(head);

printf("%s\n", temp->word);

free(temp);

}

printf("\n");

return 0;

}

// Week9Question3.h

struct Sentence {

char word[20];

struct Sentence \* next;

};

typedef struct Sentence sentence;

typedef sentence \* sent\_ptr;

sent\_ptr head = NULL;

// Push a new word to the top of the stack.

sent\_ptr pushWord(void) {

sent\_ptr newWord = (sentence \*) malloc(sizeof(sentence));

newWord->next = head;

head = newWord;

return head;

}

// Pop a word from the top of the stack.

sent\_ptr popWord(sent\_ptr popped) {

head = head->next;

return popped;

}

