**National University of Computer & Emerging Sciences, Karachi  
Fall-2023 FAST School of Computing  
Final Exam Solution** Fast

**18th December 2023, 9:00 am – 12:00 pm**

|  |  |  |
| --- | --- | --- |
| **Course Code:**CS1002 | **Course Name:** Programming Fundamentals | |
| **Instructor Name:** Mr. M. Shahzad, Dr. Farooque, Dr. Abdul Aziz, Mr. Zain, Mr. Basit, Ms. Sobia, Mr. Farooq Zaidi, Mr. M. Kariz | | |
| **Student Roll No:** | | **Section:** |

**Instructions:**

* Return the question paper and make sure to keep it inside your answer sheet.
* Read each question completely before answering it. There are 6questions and 5 pages.
* In case of any ambiguity, you may make assumptions. However, your assumption should not contradict any statement in the question paper.
* Do not write anything on the question paper (except your ID and group).

**Total Time:** 170 minutes **Max Points**: 100

**Q1: [20 min, 15 Points (5 each), CLO 1]**

1. Write on the answer sheet the output of the following programs, when they are executed. There are no compilation errors in the programs.

|  |  |
| --- | --- |
| 1, 2, 3  4, 5, 6 | {3, 2, 1};  {6, 5, 4};  {9, 8, 7}; |

|  |  |
| --- | --- |
|  |  |

**Q2: [30 min, 18 Points (6 each), CLO 2]** Considering the output given, complete the following code snippets. [Attempt on answer script]

|  |  |
| --- | --- |
| struct student {  char fname[30];  char lname[30];  int rollno;  float percentage;  };  void writeStudentToFile(const char \*filename) {  FILE \*fp;  struct student input;  // open student file for writing  fp = fopen(filename, "w");  if (fp == NULL) {  printf("\nFile opening error..\n\n");  exit(1);  }  printf("Enter \"exit\" as First Name to stop reading user input.");  while (1) {  printf("\nFirst Name: ");  scanf("%s", input.fname);  if (strcmp(input.fname, "exit") == 0)  break;  printf("Last Name : ");  scanf("%s", input.lname);  printf("Roll Number : ");  scanf("%d", &input.rollno);  printf("Percentage : ");  scanf("%f", &input.percentage);  // write student data to file  fwrite(&input, sizeof(struct student), 1, fp);  }  fclose(fp);  } | Output:  Enter "exit" as First Name to stop reading user input. First Name: Ali Last Name : Iqbal Roll Number  : 101 Percentage : 90.50  First Name: Naima Last Name : Ali Roll Number  : 102 Percentage : 95.50  First Name: exit |
| int\* getMinMax(int \*numbers, const int size) {  int i;  int min = \*numbers;  int max = \*numbers;  for (i = 1; i < size; i++) {  if (\*(numbers + i) < min)  min = \*(numbers + i);  if (\*(numbers + i) > max)  max = \*(numbers + i); }  int \*resultArray = (int\*)malloc(2 \* sizeof(int));  resultArray[0] = min;  resultArray[1] = max;  return resultArray;  } | Output:  Enter size of array: 5  Enter 5 elements in array: 1 -2 3 -1 9  Minimum value in array : -2  Maximum value in array : 9 |

|  |  |
| --- | --- |
| void removeWordFromString(char str[], char word[], char neww[]) {  int i, j = 0, k = 0, n = 0;  int flag = 0;  for (i = 0; str[i] != '\0'; i++) {  k = i;  while (str[i] == word[j]) {  i++, j++;  if (j == strlen(word)) {  flag = 1;  break;  }  }  j = 0;  if (flag == 0)  i = k;  else  flag = 0;  neww[n++] = str[i];  }  neww[n] = '\0';  } | Output:  Enter any string to remove a word from it:  Programming Fundamental  Enter the word you want to remove: gram  After removing the word from the string: Proming Fundamental |

**Q3: [25 min, 12 Points, CLO 3]**

#include <stdio.h>

#define N 10000

#define inf 1e9

int arr[N], n, x;

int foo(int sum)

{

if (sum == 0)

     return 0;

int mini = inf;

for (int i = 0; i < n; i++)

{

     if (sum - arr[i] >= 0)

     {

         int m = foo(sum - arr[i]) + 1;

         if (mini > m)

             mini = m;

     }

}

return mini;

}

int main(void)

{

scanf("%d %d", &n, &x);

for (int i = 0; i < n; i++)

     scanf("%d", &arr[i]);

printf("%d\n", foo(x));

}

**Q4: [30 min, 20 Points, CLO 2**

You need to write two functions for user authentication with encryption in C Language:

Part A. void encrypt(\*usernames, \*passwords): This function takes two pointer arrays as arguments: usernames:

An array of 100 strings containing user names, and passwords: An array of 100 strings containing passwords. Strings are null (‘\0’) terminated.

For each username and password pair, the function encrypts them using the below method:

* Each character in the string is replaced by another character that is i positions ahead in the alphabet.
* i is determined by the index of the string in the usernames array (e.g., first string element uses i=0, second element uses i=1, etc.).

Part B. int find(\*usernames, \*passwords, \*search\_username, \*search\_password): This function takes four arguments. The function searches in the encrypted usernames and passwords arrays for a matching pair corresponding to the provided search\_username and search\_password (un-encrypted). Function returns 1 if a matching username and password pair are found, 0 otherwise.

----------------------SOLUTION-----------------------

**PART 1:**

void encrypt(char \*usernames[], char \*passwords[]) {

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

char user[100];

strcpy (user, usernames[i]);

for (int j = 0; user[j] != '\0'; ++j) {

user[j] = user[j] + i;

}

usernames[i] = user;

char pw[100];

strcpy (pw, passwords[i]);

for (int j = 0; pw[j] != '\0'; ++j) {

pw[j] = pw[j] + i;

}

passwords[i] = pw;

}

}

**PART 2:**

int find(char \*usernames[], char \*passwords[], char \*search\_username, char \*search\_password)

{

int userFlag = 0;

for (int i = 0; i < 100 && userFlag != 1; i++)

{

char user[100];

strcpy (user, search\_username);

for (int j = 0; user[j] != '\0'; ++j) {

user[j] = user[j] + i;

}

char pw[100];

strcpy (pw, passwords[i]);

for (int j = 0; pw[j] != '\0'; ++j) {

pw[j] = pw[j] + i;

}

if (strcmp(user, usernames[i]) && strcmp(pw, passwords[i]))

userFlag = 1;

}

return userFlag;

}

**Q5: [35 min, 15 Points, CLO 2]**

Part A

// Structure Definitions

struct DailyConsumption {

int day;

double unitsConsumed;

};

struct BillingTier {

double rate;

double upperLimit;

};

struct ElectricityBill {

char customerName[50];

int customerID;

struct DailyConsumption dailyConsumptions[30];

};

Part B

void calculateTotalConsumption(struct ElectricityBill \*bill)

{

double totalConsumption = 0.0;

for (int i = 0; i < 30; i++)

{

totalConsumption += bill->dailyConsumptions[i].unitsConsumed;

}

// Calculate and print the total bill based on billing tiers

double totalBill = 0.0;

int tierIndex = 0;

while (totalConsumption > billingTiers[tierIndex].upperLimit && billingTiers[tierIndex].upperLimit != -1.0)

{

totalBill += billingTiers[tierIndex].upperLimit \* billingTiers[tierIndex].rate;

totalConsumption -= billingTiers[tierIndex].upperLimit;

tierIndex++;

}

totalBill += totalConsumption \* billingTiers[tierIndex].rate;

printf("Total Bill: $%.2f\n", totalBill);

}

void findUnitFrequency(struct ElectricityBill bill)

{

int frequency[30] = {0};

double units;

for (int i = 0; i < 30; i++)

{

units = bill.dailyConsumptions[i].unitsConsumed;

frequency[i] = 1;

for (int j = i + 1; j < 30; j++)

{

if (bill.dailyConsumptions[j].unitsConsumed == units)

{

frequency[i]++;

frequency[j] = -1;

}

}

if (frequency[i] != -1) {

printf("%.2lf units frequency is %d\n", units, frequency[i]);

}

}

}

void Analysis(const struct ElectricityBill \*bill) {

// Find the second highest and third lowest electricity consumption

double highestConsumption = -1.0;

double secondHighestConsumption = -1.0;

double lowestConsumption = 1.0e9;

double secondLowestConsumption = 1.0e9;

double thirdLowestConsumption = 1.0e9;

for (int i = 0; i < 30; ++i) {

double consumption = bill->dailyConsumptions[i].unitsConsumed;

// Update highest and second highest

if (consumption > highestConsumption) {

secondHighestConsumption = highestConsumption;

highestConsumption = consumption;

} else if (consumption > secondHighestConsumption && consumption < highestConsumption) {

secondHighestConsumption = consumption;

}

// Update lowest and second lowest and third lowest

if (consumption < lowestConsumption) {

thirdLowestConsumption = secondLowestConsumption;

secondLowestConsumption = lowestConsumption;

lowestConsumption = consumption;

} else if (consumption < secondLowestConsumption) {

thirdLowestConsumption = secondLowestConsumption;

secondLowestConsumption = consumption;

} else if (consumption < thirdLowestConsumption)

{

thirdLowestConsumption = consumption;

}

}

// Display the results

printf("Days with the second highest and third lowest electricity consumption:\n");

printf("Second Highest Consumption (%.2lf units):\n", secondHighestConsumption);

for (int i = 0; i < 30; ++i) {

if (bill->dailyConsumptions[i].unitsConsumed == secondHighestConsumption) {

printf("Day %d\n", bill->dailyConsumptions[i].day);

}

}

printf("Third Lowest Consumption (%.2lf units):\n", thirdLowestConsumption);

for (int i = 0; i < 30; ++i) {

if (bill->dailyConsumptions[i].unitsConsumed == thirdLowestConsumption) {

printf("Day %d\n", bill->dailyConsumptions[i].day);

}

}

}}

**Q6: [30 min, 20 Points, CLO 4]**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char \*result = NULL;

// Function to concatenate two strings dynamically

char\* concatenateStrings(char\* str1, char\* str2) {

int len1 = strlen(str1);

int len2 = strlen(str2);

int totalLen = len1 + len2 + 1;

// Check if this is the first concatenation

if (result == NULL) {

result = (char \*)malloc(totalLen \* sizeof(char));

if (result == NULL) {

perror("Memory allocation failed. Exiting...");

exit(EXIT\_FAILURE);

}

strcpy(result, str1);

} else {

// Reallocate memory for the concatenated result

result = (char \*)realloc(result, strlen(result) + totalLen \* sizeof(char) + 1);

if (result == NULL) {

perror("Memory reallocation failed. Exiting...");

exit(EXIT\_FAILURE);

}

strcat(result, str1);

}

// Concatenate the second string

strcat(result, str2);

return result;

}

int main() {

char temp[50];

char\* input1 = NULL;

char\* input2 = NULL;

char\* concatenated = NULL;

char choice;

do {

// Input two strings of varying lengths

printf("Enter the first string: ");

fgets(temp, sizeof(temp), stdin);

temp[strcspn(temp, "\n")] = '\0'; // Remove trailing newline

input1 = (char \*)malloc(strlen(temp) + 1);

strcpy(input1, temp);

printf("Enter the second string: ");

fgets(temp, sizeof(temp), stdin);

temp[strcspn(temp, "\n")] = '\0'; // Remove trailing newline

input2 = (char \*)malloc(strlen(temp) + 1);

strcpy(input2, temp);

// Concatenate the strings

concatenated = concatenateStrings(input1, input2);

// Display the original input strings and the concatenated result

printf("\nOriginal Strings:\n");

printf("String 1: %s\n", input1);

printf("String 2: %s\n", input2);

printf("\nConcatenated Result with Previous Strings:\n");

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

// Prompt user to continue or quit

printf("Enter 'Q' to quit or any other key to continue: ");

scanf(" %c", &choice);

// Free memory for the previous inputs

free(input1);

free(input2);

// Clear input buffer

while ((getchar()) != '\n');

} while (choice != 'Q' && choice != 'q');

// Free remaining memory

free(result);

return 0;

}