סמסטר א' תשע"א

משך הבחינה: שלוש שעות

אין להשתמש בחומר עזר

**פתרון בחינה בקורס תכנות ב-C**

**מרצים**: אפרת משיח, אסף זריצקי, מתי שמרת

**מתרגלים**: נוגה אורון-לוי, ידעאל ולדמן, יהודית חסון, רינת בן-אברהם, נתן מנור

# הנחיות כלליות

* את הפתרונות יש לכתוב במסגרות המסומנות לכל שאלה. במחברת הבחינה יש להשתמש כטיוטה בלבד
* בכל סעיף ניתן להשתמש בפונקציות שהתבקשתם לכתוב בסעיפים קודמים, גם אם לא פתרתם את הסעיפים הקודמים
* ניתן להשתמש בפונקציות ספריה, אלא אם נאסר במפורש
* אין צורך לבצע #include אם אתם משתמשים בפונקציות ספרייה
* בסוף המבחן נמצא דף-עזר
* נא לכתוב בכתב קריא וברור
* אם המקום המוקצה לפתרון אינו מספיק תוכלו לכתוב גם בצידו השני של הדף
* יש לרשום את מספר תעודת הזהות ואת מספר המחברת בכל עמוד
* מומלץ לקרוא כל שאלה עד סופה, על כל סעיפיה, לפני תחילת הפתרון

בהצלחה!

***לשימוש הבודקים:***

|  |  |
| --- | --- |
| תעודת זהות |  |
| מספר מחברת |  |
| שאלה 1 |  |
| שאלה 2 |  |
| שאלה 3 |  |
| ציון |  |

**שאלה מס' 1 (33 נקודות)**

**סעיף א' (7 נקודות):**

ממשו את הפונקציה readLine. הפונקציה קוראת מחרוזת תווים מהמשתמש עד לקריאת התו '\n' (ירידת שורה) ושומרת את הקלט במערך buffer, שגודלו size. שימו לב לא לחרוג מגבולות המערך בעת קריאת הקלט.

בכל מקרה הפונקציה תקרא את **כל** השורה (עד וכולל '\n'). במידה ושורת הקלט ארוכה מגודל המערך, הפונקציה תקרא **עד** **סוף השורה** אך תשמור רק את size – 1 התווים הראשונים. התו '\n' אינו מווה חלק ממחרוזת הקלט ולכן לא יופיע במחרוזת הפלט.

הפונקציה מקבלת מערך buffer אליו תקרא את הקלט ואת גודל המערך size. לאחר שהפונקציה מסתיימת המערך buffer יכיל את המחרוזת שקראתם מהמשתמש.

**void readLine(char \*buffer, int size)**

**{**

**char c;**

**int i = 0;**

**for (c = getchar(); c!= '\n'; c = getchar())**

**{**

**if (i < size - 1)**

**buffer[i++] = c;**

**}**

**buffer[i] = '\0';**

**}**

**סעיף ב' (8 נקודות):**

בסעיף זה נממש את הפונקציה isPrefix הבודקת האם מחרוזת מתחילה בתבנית מסוימת.

הפונקציה תקבל כקלט מחרוזת str ומחרוזת prefix, המייצגת תבנית, ותחזיר true (1) אם רצף התווים בתחילת המחרוזת str תואם לתבנית prefix, אחרת היא תחזיר false (0).

התבנית הינה מחרוזת המורכבת מאותיות ומספרים וכן מהתו '?' לתו זה משמעות מיוחדת והוא מציין "כל תו". ניתן להניח שהתבנית לא מכילה תווים נוספים ואין צורך לבדוק זאת. כמו כן, ניתן להניח שאורך התבנית אינו עולה על אורך המחרוזת.

לדוגמא, "dog" היא תבנית קבועה שבה התו הראשון הוא 'd', השני הוא 'o' והאחרון הוא 'g'. לעומתה התבנית "d?g" היא תבנית דומה לראשונה אלא שהתו השני מתאר תו כלשהוא. המחרוזת "digging" אינה מתחילה בתבנית "dog" אך היא כן מתחילה בתבנית "d?g".

דוגמאות:

isPrefix( "Humpty dumpty", "Hum" ) == true

isPrefix( "Humpty dumpty", "hum" ) == false

isPrefix( "Humpty dumpty", "dumpty" ) == false

isPrefix( "Humpty dumpty", "?u?pty" ) == true

isPrefix( "democracy", "d?m" ) == true

// assumes strlen(prefix) <= strlen(str)

**int isPrefix(char \*str, char \*prefix)**

**{**

**while ( \*prefix != '\0' &&**

**(\*prefix == '?' || \*prefix == \*str) )**

**{**

**str++;**

**prefix++;**

**}**

**return \*prefix == '\0';**

**}**

**סעיף ג' (8 נקודות):**

בסעיף זה הינכם נדרשים לממש באופן **רקורסיבי** את הפונקציה countOccurrences **תוך שימוש בפונקציה מסעיף ב'**. הפונקציה countOccurances סופרת כמה פעמים מופיעה תבנית בתוך מחרוזת אחרת.

דוגמאות:

countOccurances("Humpty dumpty", "u?p") == 2

countOccurances("", "a") == 0

countOccurances("mama and papa", "?a?a") == 3

שימו לב שהמופעים יכולים לחפוף כפי שניתן לראות האחרונה שבה mama ו-papa הם מופעים התואמים את התבנית, אך גם רצף התווים "ma a" מתאים לתבנית.

הערה: מימוש שאינו רקורסיבי או שאינו עושה שימוש בפונקציה מהסעיף הקודם יזכה בניקוד חלקי בלבד.

**int countOccurances(char \*text, char \*pattern)**

**{**

**if (strlen(pattern) > strlen(text))**

**return 0;**

**return (isPrefix(text, pattern) ? 1 : 0) +   
 countOccurances(text + 1, pattern);**

**}**

**סעיף ד' (10 נקודות):**

ממשו תכנית הקולטת שתי מחרוזות מן המשתמש: שורת קלט ותבנית, ומדפיסה כמה פעמים הופיעה התבנית בשורת הקלט.

אורכה המרבי של שורת הקלט הוא 100 תווים והאורך המרבי של התבנית הוא 10 תווים.

על התכנית להבטיח כי הקלט חוקי: שתי מחרוזות הקלט אינן ריקות ומחרוזת התבנית אינה ארוכה ממחרוזת הקלט. במידה והקלט אינו חוקי על התכנית להדפיס הודעת שגיאה ולסיים.

**#define MAX\_TEXT\_LEN 100**

**#define MAX\_PATTERN\_LEN 10**

**int main()**

**{**

**char text[MAX\_TEXT\_LEN], pattern[MAX\_PATTERN\_LEN];**

**printf("Enter a line of text\n");**

**readLine(text, MAX\_TEXT\_LEN);**

**if (\*text == '\0')**

**{**

**printf("Error: empty text\n");**

**return 1;**

**}**

**printf("Enter pattern\n");**

**readLine(pattern, MAX\_PATTERN\_LEN);**

**if (\*pattern == '\0')**

**{**

**printf("Error: empty pattern\n");**

**return 1;**

**}**

**if (strlen(pattern) > strlen(text))**

**{**

**printf("Error: pattern longer than line\n");**

**return 1;**

**}**

**printf("The pattern %s occurs %d times in %s\n",   
 pattern, countOccurances(text, pattern), text);**

**return 0;**

**}**

**שאלה מס' 2 (33 נקודות)**

בשאלה זו נממש קוד להצפנת "ערבול אותיות" שתתואר בפירוט בהמשך.

**סעיף א' (6 נקודות):**

נתונה הפונקציה allocateMatrix שמקצה דינמית מערך דו-מימדי של תווים, על פי המימדים שניתנו לה כפרמטרים: rows – מספר שורות, cols – מספר עמודות.

char\*\* allocateMatrix(int rows, int cols)

{

char\* data;

char\*\* table;

int i;

data = (char\*) malloc(rows \* cols \* sizeof(char));

table = (char\*\*) malloc(rows \* sizeof(char\*));

if (data == NULL || table == NULL)

{

printf("memory allocation failed");

exit(1);

}

for (i = 0; i < rows; i++) {

table[i] = data + (i \* cols);

}

return table;

}

ממשו את הפונקציה freeMatrix המשחררת מערך דו-מימדי שהוקצה דינמית על ידי הפונקציה allocateMatrix.

**void freeMatrix(char\*\* mat)**

**{**

**free(mat[0]);**

**free(mat);**

**}**

**סעיף ב' 15) נקודות):**

הצפנת *"ערבול אותיות"* היא הצפנה שבה מכניסים הודעה למטריצה דו מימדית בעלת 7 עמודות וכמות השורות המינימלית הדרושה להכיל את ההודעה. ההודעה נכתבת במטריצה שורה אחר שורה, אם השורה האחרונה לא מתמלאת, את שארית המטריצה ממלאים בתו '#' (תו זה אינו יכול להופיע במחרוזת הקלט).

ההצפנה מבוצעת על ידי קריאת האותיות מהמטריצה עמודה אחר עמודה, כמתואר באיור בהמשך.

להלן איור המדגים את סדר ההכנסה וההוצאה של התווים עבור מטריצה עם 3 שורות,

סדר הכנסת התוים:

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

סדר הוצאת התוים:

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

לדוגמא: כדי להצפין את ההודעה הבאה

"A SPY WAS CAUGHT"

נכניס את ההודעה למטריצה בגודל 3x7:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W |  | Y | P | S |  | A |
| G | U | A | C |  | S | A |
| # | # | # | # | # | T | H |

וההודעה המוצפנת שתקרא מהמטריצה היא:

"AAH STS #PC#YA# U#WG#"

בסעיף זה תממשו שתי פונקציות עזר הדרושות להצפנה:

void fillMatrix(char\*\* table, int rows, int cols,

char\* msg)

הפונקציה fillMatrix מקבלת מערך דו-מימדי (table), את מספר השורות והעמודות בו (rows, cols) ואת המחרוזת שיש להצפין. הפונקציה תמלא את המטריצה בתווי ההודעה (ובתווי '#') כמתואר למעלה. הניחו כי גודל המטריצה מתאים להודעה (כלומר rows מתאים לאורך ההודעה).

void readCode(char\*\* table, int rows, int cols,

char\* code)

הפונקציה readCode מקבלת מערך דו-מימדי (table) המכיל הודעה ואת מספר השורות והעמודות בו (rows, cols). הפרמטר code מצביע לתחילת מערך של תווים. הפונקציה תקרא מהמטריצה את ההודעה המוצפנת ותשמור אותה **כמחרוזת** במערך code. ניתן להניח שהמערך code הוא בגודל מספיק.

**void fillMatrix(char\*\* table, int rows, int cols, char\* msg)**

**{**

**int i,j,k=0;**

**int len = strlen(msg);**

**for (i = 0; i < rows; i++){**

**for (j = 0; j < cols; j++, k++){**

**if (k < len)**

**table[i][j] = msg[k];**

**else**

**table[i][j] = '#';**

**}**

**}**

**}**

**void readCode(char\*\* table, int rows, int cols, char\* code)**

**{**

**int i,j,k = 0;**

**for (j = 0; j < cols; j++){**

**for (i = 0; i < rows; i++, k++){**

**code[k] = table[i][j];**

**}**

**}**

**code[k] = '\0';**

**}**

**סעיף ג' (12 נקודות):**

כתבו פונקציה המצפינה הודעה בשיטת *"ערבול אותיות"*. להלן חתימת הפונקציה:

void encrypt(char\* msg, char\* code)

הפונקציה encrypt מקבלת מחרוזת msg אותה היא מצפינה, ואת המחרוזת המוצפנת היא כותבת למחרוזת code. תהליך ההצפנה מתבצע כפי שתואר בסעיף ב'. ניתן להניח ש-code הוא מצביע לתחילת מערך של תוים בגודל המספיק כדי להכיל את המחרוזת המוצפנת.

בפונקציה זו יש להשתמש בכל הפונקציות שהוגדרו בסעיפים הקודמים.

**#define NUM\_COLS 7**

**void encrypt(char\* msg, char\* code)**

**{**

**char\*\* table;**

**int len = strlen(msg);**

**int cols = NUM\_COLS ;**

**int rows = len/cols;**

**if (len%cols > 0)**

**rows++;**

**table = allocateMatrix(rows, cols);**

**fillMatrix(table, rows, cols, msg);**

**readCode(table, rows, cols, code);**

**freeMatrix(table);**

**}**

**שאלה מס' 3 (34 נקודות)**

בשאלה זו נעסוק במיזוג של רשימות קניות.

לצורך מימוש הרשימה נגדיר מבנה המתאר מוצר ברשימת הקניות:

typedef struct product {

char name[MAX\_NAME\_LEN];

double quantity;

struct product \*next;

} Product;

**סעיף א' (5 נקודות):**

ממשו פונקציה היוצרת מבנה חדש ומאתחלת אותו בעזרת הערכים המועברים לפונקציה. הפונקציה מקבלת את שם המוצר וכמותו ומחזיר מצביע למבנה החדש, או NULL אם הקצאת הזיכרון נכשלה.

**Product\* createProduct(char\* name, double quantity)**

**{**

**Product \*product = (Product\*) malloc(sizeof(Product));**

**if (product != NULL)**

**{**

**strcpy(product->name, name);**

**product->quantity = quantity;**

**product->next = NULL;**

**}**

**return product;**

**}**

**סעיף ב' (9 נקודות):**

בסעיף זה עליכם לממש חיפוש של מוצר ברשימת הקניות בהתבסס על שם המוצר. הפונקציה findProduct מקבלת מצביע לראש רשימת הקניות ואת שם המוצר אותו מחפשים. הפונקציה תחזיר מצביע למוצר המבוקש או NULL במקרה והמוצר לא נמצא ברשימה.

הניחו שאין מוצרים שמופיעים יותר מפעם אחת באותה רשימה.

**Product\* findProduct(Product \*head, char \*productName)**

**{**

**while (head != NULL &&**

**strcmp(head->name, productName) != 0)**

**{**

**head = head->next;**

**}**

**return head;**

**}**

**סעיף ג' (20 נקודות)**

נתונה הפונקציה addProduct המוסיפה מוצר לרשימה נתונה. הפונקציה מוסיפה את המוצר בתחילת הרשימה.

Product\* addProduct(Product \*head, char \*productName,   
 double quantity)

{

Product \*toAdd = createProduct(productName, quantity);

if (toAdd != NULL)

toAdd->next = head;

return toAdd;

}

אבא ואמא הכינו כל אחד רשימת קניות. עלינו לעזור לדני, בנם, למזג את שתי רשימות הקניות לרשימה מאוחדת אחת. על הרשימה להכיל כל מוצר בדיוק פעם אחת, כאשר הכמות ביחידות של כל מוצר ברשימה המאוחדת היא **המקסימום** בין הרשימות.

לדוגמא, אם הרשימה של אבא הכילה 2 ליטר חלב, 1 קוטג' ו-2 כיכרות לחם,

והרשימה של אמא הכילה כיכר לחם, 3 ליטר חלב, 5 תפוחים ו-3 קלמנטינות,

הרשימה המאוחדת תכיל 3 ליטר חלב, 1 קוטג', 2 כיכרות לחם, 5 תפוחים ו-3 קלמנטינות.

ממשו פונקציה המקבלת שתי רשימות קניות וממזגת את הרשימה השנייה לתוך הראשונה כמתואר למעלה. הפונקציה מחזירה מצביע לראש הרשימה המאוחדת.

הדרכה:

עברו על כל אחד מהמוצרים ברשימה השנייה, אם הוא נמצא ברשימה הראשונה – עדכנו את הכמות, אחרת, הוסיפו אותו לרשימה הראשונה.

שימו לב:

* עליכם לעדכן את הרשימה הראשונה, אסור לשנות דבר ברשימה השנייה.
* הניחו שאין מוצרים שמופיעים יותר מפעם אחת באותה רשימה.
* סדר המוצרים ברשימה הממוזגת אינו משמעותי.

ראו מקום לפתרון בעמוד הבא.

**Product\* mergeLists(Product \*list1, Product \*list2)**

**{**

**Product \*product = NULL;**

**while (list2 != NULL)**

**{**

**product = findProduct(list1, list2->name);**

**if (product != NULL)**

**{**

**if (product->quantity < list2->quantity)**

**{**

**product->quantity = list2->quantity;**

**}**

**}**

**else**

**{**

**list1 = addProduct(list1,**

**list2->name, list2->quantity);**

**if (list1 == NULL)**

**return NULL;**

**}**

**list2 = list2->next;**

**}**

**return list1;**

**}**

**דף-עזר למבחן בתכנות**

**שימו-לב**: מטרת דף-העזר היא רק לחסוך לימוד בעל-פה של ההכרזות על הפונקציות המופיעות בו. יתכן שאין צורך להשתמש בפונקציות הללו כדי לפתור את המבחן, ויתכן שצריך להשתמש בפונקציות שנלמדו אך אינן מוזכרות בדף-העזר.

פונקציות נבחרות מהספריה string.h:

|  |  |
| --- | --- |
| 1) השוואת מחרוזות | int strcmp(const char \*s1, const char \*s2) |
| 2) השוואת n תווים ראשונים של מחרוזות | |
| int strncmp(const char \*s1, const char \*s2, int n) | |
| 3) חיפוש מופע ראשון של תו במחרוזת | |
|  | char\* strchr(const char \*s, char c) |
| 4) חיפוש מופע אחרון של תו במחרוזת | char\* strrchr(const char \*s, char c) |
| 5) חיפוש מופע ראשון של מחרוזת במחרוזת אחרת | |
| char\* strstr(const char \*s, const char \*to\_find) | |
| 6) העתקת מחרוזת | char\* strcpy(char \*dest, const char \*src) |
| 7) העתקת n תווים ראשונים של מחרוזת | |
| char\* strncpy(char \*dest, const char \*src, int n) | |
| 8) שירשור מחרוזות | char\* strcat(char \*dest, const char \*src) |
| 9) שירשור n תווים ראשונים ממחרוזת | |
| char\* strncat(char \*dest, const char \*src, int n) | |
| 10) מציאת אורך מחרוזת | int strlen(const char \*s) |

פונקציות נבחרות מהספריה math.h:

|  |  |
| --- | --- |
| 1) שורש ריבועי | double sqrt(double x) |
| 2) חזקה | double pow(double base, double exponent) |
| 3) ערך מוחלט (שלמים) | int abs(int n) |
| 4) ערך מוחלט (ממשיים) | double fabs(double x) |
| 5) עיגול למעלה | double ceil(double x) |
| 6) עיגול למטה | double floor(double x) |

פונקציות נבחרות מהספריה ctype.h:

|  |  |
| --- | --- |
| 1) האם תו הוא אלפא-נומרי (a-z, A-Z, 0-9) | int isalnum(int c) |
| 2) האם תו הוא אות (a-z, A-Z) | int isalpha(int c) |
| 3) האם תו הוא סיפרה (0-9) | int isdigit(int c) |
| 4) האם תו הוא אות קטנה (a-z) | int islower(int c) |
| 5) האם תו הוא אות גדולה (A-Z) | int isupper(int c) |
| 6) האם תו הוא whitespace (רווח, \n' ' או ('\t' | int isspace(int c) |
| 7) הפיכת אות גדולה לאות קטנה | int tolower(int c) |
| 8) הפיכת אות קטנה לאות גדולה | int toupper(int c) |

פונקציות נבחרות מהספריה stdlib.h:

|  |  |
| --- | --- |
| 1) הקצאת זיכרון דינאמית | void\* malloc(unsigned int size) |
| 2) הקצאת זיכרון דינאמית ואיפוס הזיכרון שהוקצה | |
| void\* calloc(unsigned int nelements, unsigned int elementSize) | |
| 3) הקצאה מחודשת של זיכרון | void\* realloc(void \*pointer, int size) |
| 4) שחרור זיכרון שהוקצה דינאמית | void free(void \*pointer) |