פקולטה: מדעי הטבע
מחלקה: מדעי המחשב
שם הקורס: מבוא למחשבים ושפת C
קוד הקורס: 2-7028510
תאריך בחינה: **שאלות חזרה למבחן. חשוב: אין להסיק ששאלות אחרות לא יכולות להישאל במבחן, אין להסיק כי נושאים מסויימים בסיליבוס לא יכולים להופיע בבחינה ואין להסיק על אורך המבחן משאלון זה!**
משך הבחינה: שעתיים
שם המרצה: ד"ר אופיר פלא
חומר עזר: פתוח
שימוש במחשבון: לא
<u>הוראות כלליות:</u>

- הניסוח הוא בלשון זכר מטעמי נוחות ומתייחס לכולם/ן!
- אין בחירה במבחן. יש לענות על כל השאלות.
- ניתן להסתמך על כל סעיף במבחן גם אם לא פתרתם אותו על מנת לפתור סעיף אחר במבחן.
- ניתן לרשום "לא יודע/ת" על סעיף ולזכות ב20% מהנקודות המוקנות לסעיף הספציפי.
- במידה ולסעיף ניתנה תשובה ובנוסף נרשם לגבי הסעיף "לא יודע/ת" אזי הניקוד שיינתן לסעיף יהיה 0 מבלי שהתשובה תיקרא.
- אם לא רשום דבר בסעיף או שזה כלל לא נמצא ההנחה היא שנרשם "לא יודע/ת" עבור אותו סעיף.
- יורדו נקודות על פתרון נכון שאינו אופטימלי בהתייחס לנלמד בקורס.
- לכל אורך הבחינה הניחו כי (**ייתכן ובבחינה המספרים יהיו שונים בהנחה!**):

```
sizeof (char) = 1
sizeof (int) = 4
sizeof (double) = 8
sizeof (void *) = 4
```

שאלות (שימו לב הפיתרונות מופיעים אחרי הבחינה ומומלץ להשתמש בהם רק אחרי ניסיון פיתרון)

1. מתכנת כתב את התוכנית הבאה. התוכנית מתקמפלת. מה תהיה התנהגות התכנית (יש לתת פלט מדויק אם אפשר או להסביר מדוע זה לא אפשרי), הסבירו.

```
#include <stdio.h>
struct A {
      int* _p;
};
void foo(struct A* ap) {
      int arr[]= {1,2,3};
      ap->_p= arr;
      for (size_t i= 0; i<sizeof(arr)/sizeof(arr[0]); ++i) {
            printf("%d ",(ap->_p)[i]);
      }
}

int main() {
      struct A a;
      foo(&a);
      return 0;
}
```

2. להלן הקובץ q2.c:

```
// q2.c
#include <stdio.h>
#define f1 f2
int f1 () { return 1; }
int f2 () { return 2; }
int main() {
      printf("%d\n",f2());
      return 0;
}
```

מה תהיה תוצאת הפעלת הפקודות הבאות ב-shell? הסבירו.

```
gcc -Wall -c q2.c -o q2.o
gcc -Wall q2.o -o q2
./q2
```

3. מתכנת כתב את התוכנית הבאה. התוכנית מתקמפלת. מה תהיה התנהגות התכנית (יש לתת פלט מדויק אם אפשר או להסביר מדוע זה לא אפשרי)? הסבירו.

```
#include <stdio.h>
#define ARR_SIZE 3
int main () {
      char realarr[ARR_SIZE];
      char* arr;
```

```
	for (size_t i= 0; i<ARR_SIZE; ++i) {
		realarr[i]= i + '1';
	}
	arr= &(realarr[-1]);
	// %c => char format
	printf ("%c", arr[ARR_SIZE]);
	return 0;
}
```

4. בשאלה זו אתם נדרשים להשלים קובץ header אחד ולממש לו 2 אמפלמנטיציות ב-2 קבצי c שונים (version 1,2).

קיבלתם את הקובץ myString.h **החלקי** הבא:

```
// Builds a new myString object which has a copy of str
// changing str after this function won't change the returned myString
// Note: you should use myStringFree to free resources.
// Time complexity:
// version 1&2: O(strlen(str))
// sizeof(myString)
// (assuming struct is equal to the sum of its ingredients):
// version 1: sizeof(size_t) + sizeof(void*)
// version 2: sizeof(void*)
____________________  myStringAllocAndCopy(____________________ str);

// Free all resources of the given myString
// Does nothing for NULL.
____________________ myStringFree(____________________ sp);

// Returns the c string stored in the given myString
// Does not allow changing of the chars
____________________ myStringCStr(____________________ sp);

// Returns the size of the string
// Time complexity:
// version 1: O(1)
// version 2: O(myStringSize(sp))
____________________ myStringSize(____________________ sp);
```

להלן קובץ q4.c המשתמש בקובץ ה-header הנ"ל:

```
#include "myString.h"
#include "myString.h" // yes I'm including it twice...
#include <stdio.h>

int main() {
	struct myString* sp= myStringAllocAndCopy("abc");
	printf("%s\n", myStringCStr(sp));
```

```
    printf("%zu\n", myStringSize(sp)); // zu is the format for size_t
    myStringFree(sp);
    sp= NULL;
    myStringFree(sp);
    return 0;
}
```

לאחר קומפילציה ויצירת קובץ הרצה הפלט צריך להיות:

```
abc
3
```

ללא שום שגיאות או דליפות זיכרון.

א. עליכם להשלים את קובץ ה-header בצורה הטובה ביותר. שימו לב כי בפרוטוטייפ של הפונקציות חסרים הטיפוסים וערכי ההחזרה. מומלץ קודם לקרוא את סעיף ב.

ב. עליכם לכתוב 2 קבצי c שייקראו: myString1.c ו-myString2.c שיממשו את 2 הגרסאות בתיעוד (version 1,2) כך שאם נכתוב את השורות הבאות ב-shell:

gcc –c –Wall q4.c -o q4.o

gcc –c –Wall myString1.c -o myString1.o

gcc –c –Wall myString2.c -o myString2.o

gcc –Wall q4.o myString1.o –o q41

gcc –Wall q4.o myString2.o –o q42

ייווצרו 2 קבצי executable בשמות q41 ו-q42 בהתאמה ל-2 הגרסאות בתיעוד.

שימו לב כי myString.h ו-q4.c צריכים להיות זהים ל-2 הגרסאות.

הערות נוספות:

- אפשר להשתמש בפונקציות:

```
char *strcpy(char *dest, const char *src);
DESCRIPTION:
The  strcpy()  function copies the string pointed to by
src, including the terminating null byte ('\0'),
to the buffer pointed to by dest.  The strings may not overlap,
and the destination string dest must  be
large enough to receive the copy.
size_t strlen(const char *s);
DESCRIPTION:
The strlen() function calculates the length of the string s,
not including the terminating '\0' character.
```

- אין צורך להכליל את הקבצים הדרושים לשימוש בפונקציות הספרייה הסטנדרטית.
- אין צורך להעתיק את הגדרת הפונקציה וההערות מקובץ ה-h רשמו רק את שם הפונקציה שאתם מממשים.
- אם המימוש בקובץ myString2.c זהה לזה שב-myString1.c מספיק לציין זאת בלי להעתיק אותו שוב.
- אין צורך לדאוג לכישלון של הפונקציה המקצה זיכרון.

5. מתכנת כתב את התוכנית הבאה. התוכנית מתקמפלת. מה תהיה התנהגות התכנית (יש לתת פלט מדויק אם אפשר או להסביר מדוע זה לא אפשרי)? הסבירו.

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

#define R 3
#define C 2

void swapr(int a[R][C], size_t r1, size_t r2) {
      int at[C];
      memcpy(at,   a+r1, sizeof(*a));
      memcpy(a+r1, a+r2, sizeof(*a));
      memcpy(a+r2, at  , sizeof(*a));
}

int main() {
      int arr[R][C]= { {1,2}, {3,4}, {5,6} };
      swapr(arr, 0,1);
      size_t r= 0, c= 0;
      for (r= 0; r<R; ++r) {
            for (c= 0; c<C; ++c) {
                  printf("%d ",arr[r][c]);
            }
            printf("\n");
      }
      return 0;
}
```

6. א.בהתייחס לקוד הבא:

```
1. #include <stdlib.h>
2. #include <stdio.h>

3. int g;

4. int* foo() {
5.    static int* p= (int*)(&foo);
```

```
6.    p= (int*)malloc(sizeof(int));
7.    *p= 3;
8.    return p;
9. }

10. int main() {
11.   int i;
12.   g= 3;
13.   char s1[]= "slabc";
14.   const char* s2[]= {"a","bc"};
15.   int* p= foo();
16.   int* (*pf)();
17.   pf= &foo;
18.   printf ("%d",*p);
19.   return 0;
20. }
```

הטבלה הבאה מכילה **כתובות**. עליכם לציין לכל **כתובת** את המיקום שלה בזיכרון (בהתייחס לשמם ולשלב הריצה כאשר התוכנית סיימה לבצע את מספר השורה שבסוגריים): מחסנית, גלובלי, ערימה דינמית, איזור הקוד, לא מוגדר. לדוגמא הכתובת &i לאחר שורה 11 היא כתובת במחסנית.

כתובת	מספר שורה	מיקום בזיכרון
&p	5	
p	5	
p	7	
&i	11	מחסנית
&g	11	
&g	12	
&(s1[5])	13	
s2[0]	14	
s2[2]	14	

	15	p
	16	pf
	17	pf

ב. הוסיפו קוד לשחרור כל הזיכרון בתוכנית וכתבו היכן יש להוסיף קוד זה (לפי מספרי השורות).

7. בהינתן הקובץ q4.c:

```
#include <stdio.h>

unsigned int reverseCounter() {
      static unsigned int counter= 4;
      --counter;
      return counter;
}

int main() {
      unsigned int c;
      while (-1) {
            c= reverseCounter();
            printf("%d ",(int)c);
      if (c<0) break;
      }
      return 0;
}
```

הפעלנו ב-shell את הפקודות הבאות:

```
gcc -c q4.c -o q4.o
gcc q4.o -o q4
./q4
```

הקף את התוצאה הנכונה והסבר בקצרה:

א. כישלון בשלב הקמפול.

ב. כישלון בשלב ה-linkage.

ג. התוכנית תבנה כראוי ולא תדפיס כלום ותסיים את ריצתה ללא שגיאות.

ג. התוכנית תבנה כראוי ותדפיס 0 1 2 3 ותסיים את ריצתה ללא שגיאות.

ד. התוכנית תבנה כראוי ותדפיס 0 1 2 3 ואז רצף אחר של מספרים, ייתכן וגם שליליים.

ה. התוכנית תבנה כראוי ותדפיס 0 1 2 3 ואז רצף אחר של מספרים, לא ייתכן שליליים.

ו. התוכנית תבנה כראוי ותדפיס 0 1 2 3 ומאז התוכנית יכולה או לעוף על שגיאת זמן ריצה או להדפיס מספרים.

בהצלחה !

פתרון המבחן

1. התכנית תדפיס:

1 2 3

העובדה ש-a שעל המחסנית של ה-main מכיל אחרי סוף foo מצביע לאזור לא מוגדר, לא תשנה דבר כי אנו לא ניגשים לזיכרון הזה.

2. התכנית תיכשל בשלב הקומפילציה בשל הגדרה כפולה של הפונקציה f2.

3. התכנית תדפיס '3'. אמנם הכתובת שמאוחסנת ב-arr היא במקום לא מוגדר, אבל arr+ARR_SIZE היא כתובת חוקית, מכיוון שהיא בדיוק הכתובת realarr+ARR_SIZE-1 שהיא הכתובת של האיבר האחרון במערך. תתקבל גם תשובה כמו "תו ה-ascii שמופיע שני מקומות אחרי '1'".

4. א.

```c
#ifndef MY_STRING_H_
#define MY_STRING_H_

#include <string.h>

struct myString;

/// Builds a new myString object which has a copy of str
/// (changing str after this function won't change the returned
myString)
/// Note: you should use myStringFree to free resources.
/// Time complexity:
/// version 1&2: O(strlen(str))
/// sizeof(myString) (assuming struct is equal to the sum of its
ingredients):
/// version 1: sizeof(size_t) + sizeof(void*)
/// version 2: sizeof(void*)
struct myString* myStringAllocAndCopy(const char* str);

/// Free all resources of the given myString
void myStringFree(struct myString* op);

/// Returns the c string stored in the given myString
/// Does not allow changing of the chars
const char* myStringCStr(const struct myString* sp);

/// Returns the size of the string
/// Time complexity:
/// version 1: O(1)
/// version 2: O(myStringSize(sp))
size_t myStringSize(const struct myString* sp);

#endif
```

ב. myString1.c

```c
struct myString {
    char* _str;
    size_t _len;
};

struct myString* myStringAllocAndCopy(const char* str) {
    struct myString* toReturn= malloc(sizeof(struct myString));
    toReturn->_len= strlen(str);
    toReturn->_str= malloc(toReturn->_len+1);
    strcpy(toReturn->_str, str);
    return toReturn;
}

void myStringFree(struct myString* sp) {
    if (sp==NULL) return;
    free(sp->_str);
    free(sp);
}

const char* myStringCStr(const struct myString* sp) {
    return sp->_str;
}


size_t myStringSize(const struct myString* sp) {
    return sp->_len;
}
```

myString2.c

```c
struct myString {
    char* _str;
};

struct myString* myStringAllocAndCopy(const char* str) {
    struct myString* toReturn= malloc(sizeof(struct myString));
    toReturn->_str= malloc(strlen(str)+1);
    strcpy(toReturn->_str, str);
    return toReturn;
}

myStringFree, myStringCStr: כמו גרסה 1

size_t myStringSize(const struct myString* sp) {
    return strlen(sp->_str);
}
```

5. התכנית תחליף בין השורה הראשונה לשנייה ותדפיס:

3 4
1 2
5 6

הסבר: a הוא למעשה מצביע למערך של C אינטים(int). לכן הגודל של *a הוא בדיוק 2 אינטים ולכן אנו מחליפים פה שורות. אותו הדבר בדיוק לגבי אריתמטיקה של פוינטרים על a.

6. א.

מיקום בזיכרון	מספר שורה	כתובת
גלובלי	5	&p
קוד	5	p
ערימה	7	p
מחסנית	11	&i
גלובלי	11	&g
גלובלי	12	&g
מחסנית	13	&(s1[5])
קוד	14	s2[0]
לא מוגדר	14	s2[2]
ערימה דינמית	15	p
לא מוגדר	16	pf
קוד	17	pf

ב. בין שורה 18 לשורה 19 (free(p.

7. ד. משתנה unsigned int הוא אף פעם לא שלילי. אנו מטילים אותו ל-int ומדפיסים את הערך, ולכן ייתכן שנקבל מספרים שליליים בהדפסה. הערה: למעשה אם יש שימוש בכל הביטים בכל אחד מהטיפוסים מובטח שנקבל מספרים שליליים.