

סדנת תכנות ב - C

מועד ב

מרצה: אורי מוסנזון

21.04.05

הנחיות

- משך הבחינה: שעהיים.
- יש לענות על כל שאלות הבחינה.
- ניתן להשתמש בכל חומר עזר כתוב. אין להשתמש במכשירי חישוב.
- הכתיבה תתבצע כולה על דפי הבחינה.
- שאלות אמריקאיות: יש להקיף בעיגול את האות שבתחילת השורה המתאימה.
- שאלות הדורשות כתיבה: כתבו רק בשורות המיועדות לכתיבה, הן אמורות להספיק. במקרה של חריגה בשורה או שתיים, אם הסיבה היא רק סגנון כתיבה, התשובה תחשב תקינה.
- יש להקפיד על רישום ברור של מספר תעודת הזהות!

1 (6%)

נתון הקוד הבא:

```
#include <stdio.h>
#include <stdlib.h>
struct A {
    struct A *_p;
    int _data;
};
int main() {
    struct A a,*p,**p2;
    p = a._p = &a;
    p2 = (struct A**)malloc(sizeof(struct A*));
    p2[0] = &a;
    if((*p2)->_p == p->_p)
        printf("equal\n");
    else
        printf("not equal");
    free(p2);
    return 0;
}
```

כתבו מה יהיה הפלט:

תשובה:

equal

2. (5%)

בשלב שלפני ההדפסה בתוכנית, איזה מהביטויים הבאים מבטא זיכרון הנמצא על הערמה ואיזה על המחסנית?

a,p,p2,*p,*p2,**p2

תשובה:

a,p,p2,*p,*p2 על המחסנית, ו - *p2 על הערמה.

3. (5%) ציינו את הטיפוס של כל אחד מהביטויים הבאים:

```

&(p->p) ____A**____
&p2 ____A***____
&**p2 ____A*____
*&**p2 ____A____

```

4. (5%)

לגבי כל אחד מהביטויים הבאים, ציינו אם הוא חוקי ואם כן, בכמה בתים יגדל הערך (הניחו ש
 sizeof(int)=sizeof(void*)=4 ושה - sizeof של struct שווה לסכום ה - sizeof של השדות שלו):

```

p2++; ____4____
(*p2)++; ____8____
(**p2)++; ____לא חוקי____

```

5. (5%)

הקובץ f1.c מכיל את הקוד הבא:

```
void foo() {}
```

והקובץ f2.c מכיל את הקוד הבא:

```
#include "f1.c"
void foo();
```

ניסו להעביר את f2.c קומפילציה באופן הבא:

```
gcc -Wall -c f2.c
```

- א. הקומפילציה תעבור באופן תקין.
- ב. הקומפילציה לא תעבור מכיוון שאסור להכליל קבצי מימוש (.c).
- ג. הקומפילציה לא תעבור מכיוון ש - foo() מוצהרת פעמיים.
- ד. הקומפילציה תעבור, אבל תהיה בעיית linking.

תשובה: א.

6. (5%)

מה יהיה הפלט של התוכנית הבאה:

```

#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int _data;
    struct Node* _next;
} Node;

Node* createNode(int data, Node* next) {
    Node* ret = (Node*)malloc(sizeof(Node));
    ret->_data = data;
    ret->_next = next;
    return ret;
}

void printList(Node *p) {
    if(!p)
        return;
    printf("%d->",p->_data);
    printList(p->_next);
}

```

```
}
```

```
int main() {  
    Node *p = createNode(7,createNode(1,createNode(2,createNode(5,0))));  
    printList(p);  
    return 0;  
}
```

פלט:

תשובה:

7->1->2->5->

7. (5%)

ניח שהיינו מחליפים את ה- main של התוכנית האחרונה בקוד הבא:

```
int main() {  
    Node  
    *root = createNode(7,0),  
    *ls = createNode(5,root),  
    *rs = createNode(9,root),  
    *ls1 = createNode(8,rs),  
    *rs1 = createNode(10,rs);  
    ....  
}
```

- האם אפשר להתייחס למבנה הנתונים שייצרנו כמבנה נתונים של עץ בינארי?
א. כן מדובר בעץ בינארי לכל דבר, ואפשר לבצע עליו את כל הפעולות הרגילות של עץ.
ב. לא, במבנה הנתונים הנ"ל יש מעגל.
ג. לא, מכיוון שההצבעות הן מהבנים לאבא, אי אפשר לבצע את הפעולות הרגילות.
ד. כן, אבל יעילות הריצה נפגמת מאחר ומעורבים הרבה משתנים לוקליים.

תשובה: ג.

8. (5%)

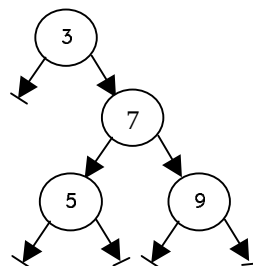
האם הפונקציה printList() של שאלה 6, כתובה היטב?

- א. כן, כתיבה רקורסיבית היא אלגנטית, קצרה ומתאימה למקרה זה.
ב. לא, עדיף להשתמש בלולאה בגלל שיקולי זמן ריצה וזיכרון מחסנית.
ג. לא, עדיף להשתמש בלולאה רק בגלל שיקולי זמן ריצה.
ד. לא, עדיף להשתמש בלולאה רק בגלל שיקולי זיכרון מחסנית.

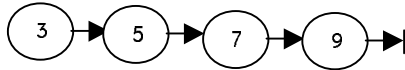
תשובה: ב.

9. (15%)

כתבו פונקציה המקבלת שורש של עץ בינארי ממין ולא ריק, ומחזירה ראש של רשימה ממוינת של קדקודיו.
אם הפונקציה מקבלת לדוגמה את העץ הבא:



עליה להחזיר את הרשימה הממוינת הבאה:



ה - struct של קודקוד יוגדר כך:

```
typedef struct node {
    int _data;
    struct node *_ls, *_rs, *_next;
} Node;
```

קודקוד זה ישמש גם לעץ וגם לרשימה. $_ls$ ו $_rs$ ישמשו כמצביעים לבנים כאשר הקדקוד משמש לעץ, ו $_next$ ישמש כמצביע לקדקוד הבא, כאשר הקדקוד משתמש לרשימה. הפונקציה שלכם תקרא tree2list ותופעל באופן הבא:

```
int main() {
    Node* root=0;
    ... bulild tree ..
    Node *p = tree2list(root);
    printList(p);
    return 0;
}
```

כאשר printList יכולה להיות ממומשת באופן הבא:

```
void printList(Node *p) {
    while(p) {
        printf("%d->", p->_data);
        p=p->_next;
    }
}
```

שימו לב שאין צורך לנתק את קשרי העץ הקיימים, רק "לתפור" את קדקודיו ע"י שימוש בשדה $_next$. מותר להשתמש בפונקציות עזר. על הקוד להיות יעיל המידה סבירה.
תשובה:

```
void tree2listRec(Node* root, Node** first, Node** last) {
    if(!root)
        return;
    *first = *last = root;
    Node *lastL=root, *firstR=0;
    tree2listRec(root->_ls, first, &lastL);
    tree2listRec(root->_rs, &firstR, last);
    lastL->_next = root;
    root->_next = firstR;
}
```

```
Node* tree2list(Node* root) {
    Node *first, *last;
    tree2listRec(root, &first, &last);
    return first;
}
```

10. (5%)

הפונקציה insert מקבלת שורש של עץ בינארי ממיון, ומכניסה לתוכו ערך חדש. היא הוגדרה באופן הבא:

```
void insert(Node* root, int data) { ... }
```

מה הבעיה ?

- א. הפונקציה אינה מחזירה ערך.
- ב. הפונקציה מקבלת int במקום Node להכנסה.
- ג. מכיוון ש insert צריכה להיות רקורסיבית, היא אמורה לקבל פרמטרים נוספים.
- ד. הפונקציה צריכה לקבל את root כמצביע למצביע.

תשובה: ד.

11. (6%)

נתונה התוכנית הבאה :

```
#include <stdio.h>
typedef void (*Func)(void*);
int count = 0;
void foo2(void* f);

void foo1(void* f) {
    printf("foo1()\n");
    ++count;
    if(f == foo1 || count == 4)
        return;
    ((Func)f)(foo1);
}

void foo2(void* f) {
    printf("foo2()\n");
    ++count;
    if(f == foo2 || count == 4)
        return;
    ((Func)f)(foo2);
}

int main() {
    foo1(foo1);
    foo1(foo2);
    return 0;
}
```

כתבו מה יהיה פלט התוכנית:
תשובה:

```
foo1()
foo1()
foo2()
foo1()
```

12. (5%)

הסבירו לשם מה היה נחוץ ה - cast בתוכנית הקודמת:

```
((Func)f)(foo1);

f(foo1);
```

ומה היה קורה אם היינו משמיטים אותו, וכותבים :

תשובה:

ללא ה - cast, התוכנית לא הייתה עוברת קומפילציה מכיוון שלא ניתן לבצע שום פעולה עם מצביע סתמי. יש צורך להמיר את המצביע הסתמי למצביע לפונקציה ורק אז ניתן להשתמש בו ככזה.

13.(7%)

נתונות הפונקציות הבאות:

```
void f1() {printf("it's one!\n"); }
void f2() {printf("it's two!\n"); }
```

כתבו תוכנות הקולטות מספר מהמשתמש, אם הוא אחד, מפעילה את f1 ואם הוא שניים היא מפעילה את f2. לתוכנית שלכם אסור להשתמש באף פקודת if. אין צורך לבדוק את תקינות הקלט:

תשובה:

```
typedef void (*FUNC) ();
```

```
int main() {
    FUNC arr[2];
    arr[0] = f1; arr[1] = f2;
    int i;
    scanf("%d",&i);
    arr[i-1]();
    return 0;
}
```

14.(4%)

השוו את טכניקת התכנות שבא השתמשתם לתכנות הרגיל הכולל if. מה היתרונות ומה החסרונות?

תשובה: בדוגמה המנוונת שניתנה, ברור שעדיף להשתמש ב - if, זה הרבה יותר פשוט. לטכניקת שהוצגה כאן יש פוטנציאל להיות מהירה יותר וגמישה יותר להרחבות ושינויים.

15.(6%)

נתון הקוד הבא:

```
#include <stdlib.h>
```

```
typedef struct A {
    struct A* _array;
} A;
```

```
int main() {
    A* array = (A*)malloc(sizeof(A)*100);
    int i;
    for(i=0; i<100; ++i) {
        array[i]._array = (A*)malloc(sizeof(A)*200);
        array[i]._array[i]._array = array;
    }
    ...free...
    return 0;
}
```

הוסיפו קוד לשחרור כל הזיכרון שהוקצה.

תשובה:

```
for(i=0; i<100; ++i)
    free(array[i]._array);
free(array);
```

16. (6%)

התבוננו בתוכנית הבאה :

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

int* foo1(int* p) {return p+1;}
int* foo2() {
    int array[10];
    int i;
    for(i=0; i<10; ++i)
        array[i] = 17;
    return foo1(array+7);
}

int main() {
    printf("%d",*(foo2()));
    return 0;
}
```

התוכנית איננה תקינה מבחינת האופן שבו היא עובדת עם הזיכרון. הסבירו מה הבעיה:
תשובה:

הפונקציה foo2 מחזירה מצביע לאבר במערך לוקאלי - array[8]. מקום זה איננו חוקי לשימוש לאחר ש foo2 מסתיימת.

17. (5%)

האם הפונקציה הבאה תקינה?

```
void foo(int n, int*p) {
    if(n<=0) {
        (*p)++;
        return;
    }
    int a;
    foo(n-1,&a);
}
```

- א. כן.
- ב. לא, היא משנה ערך במקום לא חוקי על המחסנית.
- ג. לא, היא תמיד תגרום לדליפת זיכרון.
- ד. לא, הערך של p איננו מוגדר אחרי קריאה רקורסיבית אחת.

תשובה: א.