

TEL AVIV UNIVERSITY

RAYMOND AND BEVERLY SACKLER
FACULTY OF EXACT SCIENCES
SCHOOL OF PHYSICS & ASTRONOMY



אוניברסיטת תל-אביב

הפקולטה למדעים מדוייקים
ע"ש ריימונד וברלי סאקלר
בית הספר לפיסיקה ואסטרונומיה

בחינה במחשבים לפיסיקאים, לתלמידי פיסיקה שנה א'

סמסטר ב' 2004 מועד א' תשס"ד

מרצה: ד"ר רנן ברקנא **מתרגלים:** אבי שפורר, שי מוזס,
איתי מר אור, ועדו אדם

תאריך: 20/06/2004

משך הבחינה: שלוש שעות.

בחירה: ענו על שלוש שאלות מתוך ארבע (סה"כ 100 נקודות. נקודה אחת תקבלו בחינם).

חומר עזר: מותר כל חומר עזר כתוב (כולל ספרים), אך לא מחשבון.

הערה שימושית: ערכו המספרי ב-C של התו 'a' הוא 97, של 'b' הוא 98 וכך הלאה.

1. קידוד.

א. כתבו פונקציה `void encode(char *word)` העושה קידוד פשוט של מילה ומדפיסה את הקידוד. בהינתן מילה (מחרוזת), לכל תו במילה על הפונקציה להדפיס את התו ואת מספר התווים הזוהים הנמצאים ברצף.

לדוגמא, עבור המילה: `aaabbcddaaec`
הפונקציה תדפיס: `a3b2c1d2a2e1c1`

ניתן להיעזר בפונקציה `int strlen(char *word)` המקבלת מילה ומחזירה את מספר התווים בה, כלומר, את אורכה.

(17 נקודות)

ב. כתבו פונקציה `void min_max_char(char *word)` המקבלת מילה (מחרוזת) ומוצאת את התו המופיע הכי הרבה פעמים במילה וכן את התו המופיע הכי מעט פעמים במילה. על הפונקציה להדפיס למסך את שני התווים הללו וכן את מספר הפעמים בו הם מופיעים במילה. אם יש יותר מתו יחיד המופיע הכי הרבה (או הכי מעט) פעמים, על הפונקציה להדפיס אחד מהם.

לדוגמא, עבור המילה: `abaacbbbaaec`
הפונקציה תדפיס:

`max char: a 5`
`min char: e 1`

(16 נקודות)

```

/* Solution for 2004, Moed A, question 1.a. */

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

void encode(char *word) {

    int len,i,c_num;
    char c;

    len = strlen(word); /* Number of characters without the '\0' */

    i=0;
    while (i<len) { /* External while loop */

        c = word[i];
        c_num = 0;

        /* While the next character is identical to c promote
           the index i and counter c_num. Note that in the first
           iteration of the internal while loop c will always
           equal word[i] because i hasn't been advanced yet.
           This is why c_num is initialized to zero (and not one) */
        while (word[i] == c) { /* Internal while loop */
            i++;
            c_num++;
        } /* Internal while loop */

        /* The internal while loop ends when word[i] != c, meaning the
           sequence of c characters has ended. So, we print the character
           and the length of the sequence before the external while loop
           moves on to the next character
           */
        printf("%d%c",c_num,c);

    } /* External while loop */

    printf("\n");
    /* This is only to have a nice looking output, not mandatory */

} /* encode */

```

```

/* Solution for 2004, Moed A, question 1.b. */

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

void min_max_char(char *word) {

    int len,i,j,c_num,min,max;
    char c,max_c,min_c;

    len = strlen(word); /* Number of characters without the '\0' */

    /* Initializing min to be higher than the maximal possible
       minimum value */
    min = len+1;
    /* Initializing max to be smaller than the minimal
       possible maximum value */
    max = 0;

    /* The external for loop finds appearances number of each
       character in the string and updates min and max appropriately
    */
    for(i=0; i < len; i++) { /* External for loop */
        c = word[i];
        c_num = 0;

        /* The internal for loop scans the whole string and counts the
           appearances of the character c */
        for(j=0; j < len; j++) { /* Internal for loop */

            if (c == word[j])
                c_num++;

        } /* Internal for loop */
        /* Now, c_num equals the number of appearances of the
           character c */

        /* Check if c appeared more than the current maximum
           appearing character. If so, replace it with the character c
        */
        if (c_num > max) {
            max = c_num;
            max_c = c;
        }

        /* Check if c appeared less than the current minimum
           appearing character. If so, replace it with the character c
        */
        if (c_num < min) {
            min = c_num;
            min_c = c;
        }

    } /* External for loop */

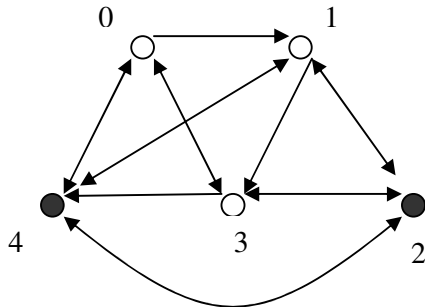
    /* Print results */
    printf("max char: %c %d\n",max_c,max);
    printf("min char: %c %d\n",min_c,min);

} /* min_max_char */

```

2. נמלה ברשת.

האלמנט הבסיסי ברשת הוא מרכז, שממנו יוצאים שלושה חיבורים חד-כיווניים החוצה. חיבורים אלו ניתן לחבר למרכזים של אלמנטים אחרים מאותו הסוג. הרשת מתוארת ע"י קובץ, שכל שורה בו מתארת אלמנט כזה ומכילה שלושה מס' שלמים המתארים את האינדקסים של האלמנטים שאליהם מחוברים שלושת החיבורים וכן 0 או 1 על מנת לציין אם במרכז האלמנט יש פירור לחם (1) או לא (0). לדוגמה,



1 3 4 0
3 2 4 0
1 3 4 1
4 2 0 0
0 1 2 1

כאשר רשימה זו היא לפי הסדר (שורה ראשונה: אלמנט 0, וכולי), ובציר עיגול מלא מסמן פירור לחם.

א. כתבו פונקציה

```
int read_network(struct node net[], int n)
```

הקוראת את נתוני הרשת מקובץ בשם network לתוך המערך net, שגודלו n. על הפונקציה להחזיר את מס' האלמנטים ברשת שנקראה. אין צורך לבדוק את תקינות הנתונים פרט לכך שאין לקרוא יותר מ-n אלמנטים.

node מוגדר ע"י:

```
struct node {
    int links[3];
    int bread;
};
```

(11 נקודות)

ב. נמלה מוצבת באלמנט מס' 0 ועוברת אקראית מאלמנט לאלמנט בעזרת החיבורים החד-כיווניים (ניתן לנוע רק בכיוון החיבור). כאשר היא מגיעה לאלמנט עם פירור לחם, הפירור נאכל, והאלמנט הופך להיות ריק מפירורים. כתבו פונקציה:

```
int walk(struct node net[], int steps)
```

המקבלת את המערך המתאר את הרשת net ומחזירה את מס' הפירורים, שהנמלה אכלה ב-steps צעדים. הניחו שאין ב-net חיבורים לאלמנטים שאינם קיימים.

הניחו שנתונה לכם הפונקציה rand48(), המחזירה מס' אקראי (double) בין 0. ל-1. (לא כולל 1).

(11 נקודות)

ג. כתבו תוכנית הקוראת רשת מהקובץ network, מקבלת מהקלט הסטנדרטי את מס' הניסויים של הצבת הנמלה על הרשת (ההצבה היא כמו בסעיף ב') ומציגה את המס' הממוצע של פירורי הלחם שנאכלו ב-20 צעדים. אתם רשאים להניח, כי ברשת אין יותר מ-100 אלמנטים.

(11 נקודות)

```

/* A solution for the random walking ant on a network given in the
   first term exam of Computers for Physicists in 2004. */

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

/* Defines */
#define LINKS 3
#define INPUT_FILE "network"
#define MAX_NODES 100
#define STEPS 20

struct node
{
    int links[LINKS]; /* Unidirectional links to three nodes */
    int bread; /* Whether there is bread in the node */
};

/* read_network() reads the network specification from a file
   called network.
   Arguments:
   net - an array in which the network specs are to be stored
   n - the size of the array
   Returns the number of nodes read. */

int read_network(struct node net[], int n)
{
    int nodes=0; /* the number of nodes read */
    FILE *infile;
    /* Open the file */
    if ((infile=fopen(INPUT_FILE, "r"))==NULL)
        return 0; /* No nodes read */
    /* Read the network specifications */
    while (nodes<n &&
           fscanf(infile, "%d %d %d %d", &net[nodes].links[0],
                 &net[nodes].links[1], &net[nodes].links[2],
                 &net[nodes].bread)!=EOF)
        nodes++;
    fclose(infile);
    return nodes; /* Return the number of nodes read */
}

/* walk() performs a random walk starting with node zero.
   Arguments:
   net - the network specification array
   steps - the number of steps to perform
   Returns: the number of bread crumbs found by the ant. */

int walk(struct node net[], int steps)
{
    int bread_crumbs=0; /* No. of bread crumbs */
    int cur_node=0; /* the current node */
    int s;
    /* Perform steps+1 steps */
    for (s=0; s<steps+1; s++)
    {
        if (net[cur_node].bread==1)
        {
            /* Found bread */
            bread_crumbs++;
            net[cur_node].bread=-1; /* Mark as "eaten" */
        }
    }
}

```

```

    }
    /* Move on */
    cur_node=net[cur_node].links[(int)(LINKS*drand48())];
}
return bread_crumbs;
}

void main()
{
    struct node net[MAX_NODES]; /* the network */
    int nodes; /* the number of nodes in the network */
    int crumbs=0; /* the number of crumbs eaten */
    int experiments; /* the number of experiments performed */
    int i, j;
    printf("Enter the no. of experiments: ");
    scanf("%d", &experiments);
    if ((nodes=read_network(net, MAX_NODES))==0)
    {
        fprintf(stderr, "Unable to open %s\n", INPUT_FILE);
        exit(1);
    }
    for (i=0; i<experiments; i++)
    {
        /* Restore "eaten" bread crumbs */
        for (j=0; j<nodes; j++)
            if (net[j].bread===-1)
                net[j].bread=1;
        crumbs+=walk(net, STEPS);
    }
    /* Print the average */
    printf("The average no. of crumbs eaten is %f\n",
        (float)crumbs/experiments);
}

```

Notes:

- 1) Note that **nodes<n** appears *before* the **fscanf()** call. This is rather essential since the evaluation of logical expressions is stopped as soon as the final outcome of the expression is found. In this case, if **nodes<n** is false (i.e., **nodes** is equal to **n**), the **fscanf()** function is not executed so no attempt is made to store data past the array's valid index range and only the first **n** elements are read.
- 2) Testing the value of the file pointer is usually a good idea, though it was not required according to the question.
- 3) As soon as the loop in **read_network()** detects that the end of the input file has been reached, it terminates. Thus the number of elements truly read is returned correctly.
- 4) The function **walk()** performs **steps+1** steps so crumbs both in the first element and the last element reached are eaten.
- 5) If the variable **crumbs** were not cast into **float**, the average would be rounded to the next lower integer value.

3. פולינומים ב C.

פולינום במשתנה אחד, ממעלה N, מוגדר להיות:

$$F(x) = \sum_{i=0}^N A_i x^i$$

ניתן לייצג פולינום ב C כמערך poly, שבו poly[i] מכיל את A_i , כלומר את המקדם של החזקה ה-i. נגדיר:

```
struct poly{
    int degree;          /* the polynomial degree*/
    double a[MAX_DEG]; /* polynomial coefficients*/
};
```

כאשר MAX_DEG יהיה הדרגה המקסימאלית שנרשה לייצג.

א. כתבו פונקציה המחשבת ערך פולינום בנקודה נתונה:
`double eval_poly(struct poly f, double x);`

(11 נקודות)

ב. כתבו פונקציה שגוזרת את הפולינום. מכיר:

$$F'(x) = \sum_{i=1}^N i A_i x^{i-1}$$

```
void deriv_poly(struct poly f, struct poly *df);
```

הנגזרת תשמר ב df.

(11 נקודות)

ג. כתבו פונקציה אשר מחשבת את המכפלה של שני פולינומים. ניתן (אבל לא חובה!) להיעזר בנוסחאות הבאות:

$$F(x) = \sum_{i=0}^N A_i x^i, G(x) = \sum_{i=0}^M B_i x^i$$

$$F(x) \cdot G(x) = \left(\sum_{i=0}^N A_i x^i \right) \cdot \left(\sum_{j=0}^M B_j x^j \right) = \sum_{k=0}^{N+M} \sum_{i=0}^N A_i \tilde{B}_{k-i} x^k$$

$$\tilde{B}_l = B_l \text{ if } 0 \leq l \leq M, 0 \text{ otherwise}$$

```
struct poly mult_poly(struct poly f, struct poly g);
```

הערה: ניתן להניח שהמעלה של המכפלה לא תעבור את MAX_DEG.

(11 נקודות)


```

#define MAX_DEG 7777

struct poly {
    int degree;
    double a[MAX_DEG];
};

double eval_poly(struct poly f, double x)
{
    double val,
        cur_pow;           /* will hold current x power */
    int i;

    cur_pow = 1;
    val = 0.0;

    for (i = 0; i <= f.degree; i++ ) {
        val += f.a[i]*cur_pow;
        cur_pow = cur_pow*x;      /* yes, you could use pow() instead */
    }

    return val;
}

void derive_poly(struct poly f, struct poly *df)
{
    int i;

    if ( f.degree == 0 ) {      /* f is a const polynomial */

        df->degree = 0;
        df->a[0] = 0;
        return;
    }

    df->degree = f.degree - 1;
    for (i = 1; i <= f.degree; i++)
        df->a[i-1]=i*f.a[i];

    return;
}

struct poly mult_poly(struct poly f, struct poly g)
{
    struct poly fg;
    int k, i;
    int M,N;

    N = f.degree;
    M = g.degree;

    fg.degree = M+N;

    for (k = 0; k <= M+N; k++) {

        fg.a[k] = 0;
        for (i = 0; i <= N; i++) {

            if ( (k-i >= 0) && (k-i<=M) )
                fg.a[k] += f.a[i]*g.a[k-i];
        }
    }
}

```

```
}  
  
    return fg;  
}
```

Common Mistakes (and their price in this question):

1) Syntax errors were priced at -2, no matter how many times the same error appeared.

For section (a):

Incorrect algorithm -1/-2pts

Wrong types -2pts

For section (b):

Incorrect handling of the case **f.degree=0** -1/-2pts

Incorrect algorithm (e.g., wrong formula for the derivative coefficients) -2pts

Some of you didn't understand what we wanted and tried to compute the value of the derivative at x. -7...-11pts

For section (c):

Incorrect algorithm (e.g. wrong formula for the new coefficients)

-2 pts

Using uninitialized variables (in case this leads to a real error)

-2 pts

4. נתונה התכנית הבאה:

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

void mydouble(int *x) {
    *x *=2;
}

main() {
    int matrix[4][3];
    int i,j,k=0;

    for (i=0; i<4; i+=2) {
        for (j=0; j<3; j++) {
            matrix[ i ][ j ] = k;
            matrix[ i+1 ][ 2-j ] = k/2;
            k+=3;
        }
    }

    mydouble(&matrix[1][1]+4);
    mydouble(&j);
    mydouble(&k);

    matrix[0][0] = j;
    matrix[3][2] = k;
}
```

א. שרטטו מטריצה במימדים של matrix וכתבו מהם ערכיה של matrix בסיום הריצה.

(20 נקודות)

ב. כתבו פונקציה `void print_array (int *arr, int rows, int cols)` המדפיסה את איברי המערך arr אחד אחרי השני באופן הבא:

1. בכל שורה מודפסים cols איברים מופרדים בפסיקים (ללא פסיק אחרי האיבר האחרון).
2. סה"כ מודפסות rows שורות.

ניתן להניח כי קיימים מספיק איברים במערך arr, כנדרש.

(10 נקודות)

ג. השתמשו בפונקציה שכתבתם כדי להוסיף לתכנית הנתונה למעלה שורת קוד אחת שתדפיס את matrix בסוף התכנית.

(3 נקודות)

פתרון:

א. המטריצה אחרי הלולאה הכפולה:

0	3	6
3	1	0
9	12	15
7	6	4

המטריצה אחרי כל הפעולות:

6	3	6
3	1	0
9	12	30
7	6	36

ב.

```
void print_array(int *arr, int rows, int cols)
{
    int i, j;
    for (i=0; i<rows; i++)
    {
        for (j=0; j<cols; j++)
        {
            if (j == 0)
                printf("%d", *(arr+i*cols+j));
            else
                printf(",%d", *(arr+i*cols+j));
        }
        printf("\n");
    }
}
```

כאן הייתה שגיאה נפוצה: שימוש שגוי במשתנה arr, שהוא מערך חד-מימדי ולא דו-מימדי (-4 נקודות).

ג.

התשובה הנכונה:

```
print_array(*matrix, 4, 3);
```

המשתנה matrix הוא מצביע כפול, ולכן זוהי שגיאה לשלוח אותו לפונקציה ללא *. אך היים נחמדים ולא הורדנו נקודות על שגיאה זו, שרק גורמת לאזהרה מהקומפיילר (אך הקוד עדיין רץ נכון).