גירסה 1.00 - 29.8.2002
גירסה 1.12 - 21.7.2006

עבודה עם קבצים בשפת C

ניר אדר ו-Avidor93

ממסמך זה הורד מהאתר http://www.underwar.co.il.

Nir Adar
Email: underwar@hotmail.com
Home Page: http://underwar.livedns.co.il

Avidor93
Email: avidor93@gmail.com

אנא שלחו תיקונים והערות אל המחברים.

במסמך זה נסקור כיצד עובדים עם קבצים בשפת c.
נתמקד בעבודה עם שני סוגים של קבצים: טקסט ובינארי.
נתייחס אל הדרך שבה c מבצעת את העבודה בתור "קופסה שחורה" - לא נתעניין איך מנגנון העבודה עם הקבצים עובד, אלא איך משתמשים בקבצים.

קבצים - הכרות ראשונית, קבצי טקסט

כאשר אנו כותבים תוכנית ויוצרים בה משתנים, המשתנים נשמרים בזיכרון המהיר של המחשב **(RAM)**.
כאשר התוכנית תסתיים, כל המשתנים עם תכולתם ייעלמו, וכאשר נריץ שוב את התוכנית, היא תיצור משתנים חדשים משלה.
לעתים נרצה דרך לשמור מידע - כך שנוכל להחזירו גם בריצות הבאות של התוכנית.
למשל - לא היינו רוצים שבכל פעם שאנו רוצים לכתוב מסמך ב-**Word**, אנו צריכים לכתוב אותו מהתחלה, כי לא היה ניתן לשמור את העבודה של הפעם הקודמת.
נרצה במקרה זה לשמור כל פעם את המסמך שאנו עובדים עליו, ובפעם הבאה לאחזר אותו אל **Word**, ולהמשיך לעבוד עליו.
לשם כך, קיימים רכיבי זיכרון קבועים, כגון **Hard-Disk** או דיסקטים, שמסוגלים לשמור מידע לאורך זמן, ולהביאו לפי דרישה.
המידע נשמר על התקנים אלו בקבצים, שמזוהים על ידי שמם ועל ידי הספרייה בה הם נמצאים.
שפת C מאפשרת לנו לקרוא ולכתוב קבצים. כדי לקרוא או לכתוב קובץ, אנו יוצרים מצביע מסוג **FILE**, ואז משתמשים במגוון פונקציות כדי לבצע את הפעולה הרצויה.
למתעניינים - **FILE** הוא למעשה **struct** המוגדר בשפת C, והוא בצירוף הפונקציות השונות הפועלות עליו הם **ADT** המיועד לעבודה עם קבצים. עם זאת, אין צורך לדעת עובדה זו על מנת לעבוד עם קבצים.
בשפת C יש שני סוגי קבצים לאחסון נתונים:
קובץ טקסט **(text file)** וקובץ בינארי **(binary file)**.

בשפת C פקודות קלט ופלט שונות - כגון **gets()**, **scanf()**, **printf()** ועוד.
לרוב פקודות אלו יש פקודה מקבילה, ששמה מתחיל באות **f**, שמיועדת לעבודה עם קבצים. למשל, הפקודה **fprintf()** היא פקודה שהתחביר שלה דומה (עם הבדל מסוים אחד שנראה מיד) ל-**printf()**, אולם היא מיועדת לכתיבה לקבצים, ולא אל הפלט הסטנדרטי.
קבצי טקסט:

נתחיל בדוגמא, ונביט בתוכנית הבאה הכותבת את המספרים 1 עד 10 לתוך קובץ, כל אחד מהם בשורה נפרדת.

```
#include <stdio.h>

int main()
{
     FILE *file_output;
     int iCounter;
     file_output = fopen("numbers.txt", "wt");

     for (iCounter = 1; iCounter <= 10; ++iCounter)
     {
          fprintf(file_output, "%d\n", iCounter);
     }

     fclose(file_output);
     return 0;
}
```

על מנת שנוכל להשתמש ב- `fopen()`, `fprintf()`, `FILE()` ו- `fclose()` עלינו להוסיף את `stdio.h` (כלומר, `standart input-output`).
בשורה הראשונה יצרנו מצביע ל-`FILE`. תמיד כאשר נרצה לעבוד עם קובץ ניצור מצביע מסוג זה.
הפקודה `fopen()` מקבלת שתי מחרוזות בתור פרמטרים.
המחרוזת הראשונה היא שם הקובץ שאותו אנו רוצים לפתוח, והמחרוזת השניה מציינת את האופן בו אנו רוצים לפתוח את הקובץ. בדוגמא שם הקובץ היה "`numbers.txt`" והאופן בו פתחנו אותו הוא "`wt`", כלומר `writetext` - פתחנו את הקובץ לכתיבה במצב טקסט.
הפונקציה `fopen()` מחזירה מצביע מסוג `FILE`, או `NULL` במקרה שהייתה בעיה לפתוח את הקובץ (למשל, ניסינו ליצור קובץ חדש בספרייה שאין לנו בה גישת כתיבה, או למשל ניסינו לפתוח קובץ לקריאה, אך הקובץ איננו קיים).
בטבלה הבאה נרכז את אפשרויות הגישה לקובץ:

משמעות	תווים
קריאה בלבד (יש צורך בקובץ קיים).	`r`
כתיבה בלבד (אם קיים קובץ הוא ימחק).	`w`
הוספה לקובץ קיים, או יצירת קובץ חדש.	`a`
קריאה וכתיבה לקובץ (יש צורך בקובץ קיים).	`+r`
קריאה וכתיבה לקובץ (אם קיים כבר קובץ הוא ימחק).	`+w`
הוספה לקובץ ואפשרות קריאה ממנו.	`+a`

נשפר את התוכנית שהרגע הצגנו, ונבדוק את הערך ש-`fopen()` מחזירה:

```
#include <stdio.h>

int main()
{
     FILE *file_output;
     int iCounter;

     file_output = fopen("numbers.txt", "wt");
     if (file_output == NULL)
     {
          printf("ERROR: Cannot create output file.\n");
          return 1;
     }

     for (iCounter = 1; iCounter <= 10; ++iCounter)
     {
          fprintf(file_output, "%d\n", iCounter);
     }

     fclose(file_output);
     return 0;
}
```

בלולאת ה-**for**, אנו רוצים על המספרים מ1- עד 10, ומשתמשים ב-**fprintf()** על מנת לכתוב אותם לקובץ. המבנה של **fprintf()** זהה לזה של **printf()**, מלבד העובדה שהפרמטר הראשון שלה הוא מצביע לקובץ, אליו אנו רוצים לכתוב את הנתונים.

גם לפונקצית הפלט **puts()** יש מקבילה המיועדת לכתיבה לקבצים. הפונקציה המקבילה הינה **fputs()**, וההצהרה עליה היא:

```
int fputs( const char *string, FILE *stream );
```

במקרה ואנו משתמשים בפונקציה זו, נכתוב קודם את המחרוזת שאנו רוצים לכתוב אל הקובץ, ולאחר מכן ניתן את המצביע עצמו.

נשים לב שבכל הפונקציות הנ"ל אנו מעבירים מצביע לקובץ ולא את שם הקובץ. למשל, בדוגמא, אם היינו כותבים את השורה הבאה, היינו מקבלים טעות קומפילציה:

```
fprintf("numbers.txt", "%d\n", iCounter);
```

הפרמטר הראשון חייב להיות מצביע ואיננו יכול להיות מחרוזת.

בסוף העבודה, עלינו לשחרר את המצביע לקובץ.
אנו עושים זאת על ידי הפקודה **fclose()**, המקבלת כפרמטר מצביע לקובץ.
חשוב לשחרר את המצביע לקובץ. לא תמיד כל הנתונים שאנו כותבים לקובץ נכתבים ישר על הכונן, לעיתים הם נכתבים בחוצץ זמני. כאשר אנו משתמשים בפקודה **fclose()**, אנו כותבים את הנתונים לכונן, וכן משחררים את הזיכרון שהוקצה עבור ה-**FILE**.
על מנת לא לשכוח לסגור את הקובץ, רצוי לכתוב בסוף התוכנית את הפקודה **fclose()**, ברגע שאנו פותחים קובץ כלשהו עם **fopen()**.

כעת נרצה לכתוב תוכנית שניה, שתקרא את הנתונים שהתוכנית הראשונה כתבה לקובץ, תשמור אותם במערך ותדפיס אותם.
על מנת לפתוח קובץ לקריאה, נעביר ל-`fopen()` כפרמטר ראשון את שם הקובץ, וכפרמטר שני נעביר לה "`rt`", כלומר `readtext`.

```
#include <stdio.h>

#define MAX_SIZE 10

int main()
{
    FILE *fin;
    int iCounter;
    int numbers[MAX_SIZE];

    fin = fopen("numbers.txt", "rt");
    if (fin == NULL)
    {
        printf("ERROR: Cannot open input file.\n");
        return 1;
    }

    for (iCounter = 0; iCounter < MAX_SIZE; ++iCounter)
    {
        fscanf(fin, "%d\n", &numbers[iCounter]);
        printf("%d\n", numbers[iCounter]);
    }

    fclose(fin);
    return 0;
}
```

אנו יוצרים מצביע בשם `fin`, פותחים בעזרת `fopen()` את הקובץ לקריאה.
לאחר מכן, בעזרת לולאה שרצה 10 פעמים, אנו קוראים את הנתונים מהקובץ אל מערך, ומיד לאחר מכן מדפיסים אותם.
`fscanf()` היא הפקודה המקבילה של `scanf()`. הפקודה קוראת נתונים מתוך קובץ, במקום מהקלט הסטנדרטי.

נשפר כעת את התוכנית שהצגנו בהדרגה.
ראשית, הנחנו בתוכנית כי בקובץ ישנם רק עשרה מספרים.
אולם, מה אם בקובץ ישנו מספר כלשהו של מספרים, לאו דווקא 10?

נרצה לשנות את הלולאה כך שנקרא נתונים מהקובץ כל עוד ישנם נתונים.

נניח כי כמות המספרים בקובץ קטנה מ-**MAX_SIZE**, כך שלא מתרחשת דליפת זיכרון.

נשתמש בערך המוחזר על ידי **fscanf()** בדומה לפונקציה **scanf()**, הפונקציה **fscanf()** מחזירה את מספר האיברים שהיא הצליחה לקרוא.
כאשר פעולת הקריאה תיכשל, ערכה המוחזר של הפונקציה יהיה **EOF**.

```
#include <stdio.h>

#define MAX_SIZE 10

int main()
{
    FILE *fin;
    int iCounter;
    int numbers[MAX_SIZE];

    fin = fopen("numbers.txt", "rt");
    if (fin == NULL)
    {
        printf("ERROR: Cannot open input file.\n");
        return 1;
    }

    while (fscanf(fin, "%d\n", &numbers[iCounter]) > 0)
    {
        printf("%d\n", numbers[iCounter++]);
    }

    fclose(fin);
    return 0;
}
```

כעת התוכנית תקרא מספרים מהקובץ אל המערך, כל עוד קיימים מספרים בקובץ.

בדוגמא הבאה נציג פונקצית קלט חדשה: **fgets()**, הקוראת מחרוזת מקובץ.
ההצהרה על **fgets()** היא כלהלן:

```
char *fgets( char *string, int n, FILE *stream );
```

הפרמטר הראשון ש-**fgets()** מקבלת היא מחרוזת אליה יכתבו הנתונים שיקראו מהקובץ. הפרמטר השני הוא מספר התווים המקסימלי שייקרא מהקובץ.
כך, כאשר ניצור מחרוזת באורך 80, ערך הפרמטר יהיה 80.
הפרמטר השלישי הוא מצביע לקובץ, ממנו אנו רוצים לקרוא נתונים.

פעולתה של `fgets()`: הפונקציה קוראת נתונים מהקובץ הנתון כפרמטר, לתוך המחרוזת `string`. הפונקציה מפסיקה לקרוא נתונים מהקובץ, כאשר מתקיים אחד משלושת התנאים הבאים:

- הגענו לסוף הקובץ.
- נקראו `n` תווים מהקובץ.
- נקרא התו `'\n'`.

כאשר אחד משלושת התנאים יתקיים, הפונקציה תחזיר `NULL`.

לפיכך, הפונקציה `fgets()` שימושית במיוחד כאשר יש לנו קובץ בעל מספר שורות, ואנו רוצים לנתח את השורות אחת אחרי השניה. בכל קריאה ל-`fgets()` נקרא שורה אחת, ונוכל לבצע עליה פעולות שונות.

כאשר נקרא שורה מקובץ בעזרת הפונקציה `fgets()` , והשורה תהיה ארוכה יותר ממספר התווים המקסימלי שהגדרנו, רק מספר התווים המקסימלי שהגדרנו יקרא למחרוזת, והשאר יקרא בקריאה הבאה.

נביט בדוגמא הבאה המדגימה את השימוש ב-`fgets()`. אנו קוראים נתונים מהקובץ `input.txt` ומדפיסים אותם על המסך.

```
#include <stdio.h>

#define MAX_LINE 80

int main()
{
    FILE *fin;
    char line[MAX_LINE];

    if ((fin = fopen("input.txt", "rt")) == NULL)
    {
        printf("ERROR: Cannot open input file.\n");
        return 1;
    }
    while (fgets(line, MAX_LINE, fin) != NULL)
    {
        printf("%s", line);
    }
    fclose(fin);
    return 0;
}
```

בזאת סיימנו להציג את העבודה עם קבצי טקסט ב-C.
בדפים הבאים יוצג איך לעבוד עם קבצים בינאריים ב C.

קבצים בינאריים:

כעת נעסוק בקבצים בינאריים. אנו צריכים להבין מהו קובץ בינארי.
בקובץ בינארי כל בית אינו חייב להיות תו, אלא הוא מייצג רצף של סיביות אשר מאוגדות בבתים – הנתונים בו נשמרים בדיוק כפי שהם בזיכרון.

בקובץ בינארי ניתן לקרוא רצף של בתים מהזיכרון אל הקובץ ולהיפך.
בהוראה אחת ניתן לכתוב לקובץ מערך של 100 מבנים לקובץ, או לקרוא לזיכרון מערך של 200 איברים מסוג **LONG**.

לכל קובץ בינארי יהיה פורמט שמגדיר כיצד הוא בנוי. כל מתכנת קובע פורמט עבור הנתונים שהוא צריך לשמור. עם זאת, יש מספר פורמטים שהוגדרו כתקניים למטרות שונות, כמו: bmp,mp3,jpg וכו'.

כעת נראה דוגמא לקובץ בינארי:

```
1 42 69 6e 61 72 79
```

לכאורה, אי אפשר להבין מה יש בקובץ. כדי לקרוא אותו – עלינו להבין את הפורמט שלו:

בית ראשון – מספר מחרוזת.
כל השאר באותה שורה – המחרוזת.

כאשר נקרא את הקובץ לזיכרון ונציג אותו:
1Binary.

השימוש בקבצים בינאריים שונה מהשימוש בקבצי טקסט רק בפתיחה ובפקודות.
כדי לפתוח קובץ בינארי:

```
FILE *f;
f=fopen("Binary.dat","ג'ישה+b");
```

לדוגמא:
פתיחת קובץ בינארי לכתיבה וקריאה:

```
f=fopen("Binary.dat","w+b");
```

בדיקת הפתיחה של הקובץ נעשית בדיוק כמו בקבצי טקסט.

כתיבה לקובץ בינארי:

הכתיבה לקובץ בינארי נעשית ע"י הפונקציה **fwrite()**
היא מאפשרת לכתוב קטע זיכרון שלם אל קובץ. הוא מועתק בדיוק כמו שהוא בזיכרון.
כך נוכל לכתוב מערך שלם של איברים בהוראה אחת, ולקרוא אותו מאוחר יותר כשנרצה.

הפונקציה **fwrite();**

```
fwrite(כתובת בזיכרון , גודל כל איבר , מספר איברים , קובץ);
```

כתובת הזיכרון – הכתובת של התחלת קטע הזיכרון שברצוננו לכתוב לקובץ.
כדי לציין זיכרון נשתמש ב **&** שאומר הכתובת של –
לדוגמא: **&numbers** .

גודל כל איבר הוא הגודל של האיבר שכותבים לקובץ.
נשתמש בפקודה **sizeof** כדי לחשב את הגודל:
sizeof(int) .

מספר איברים – מספר האיברים שיכתבו מכתובת הזיכרון.
לדוגמא:
אם אנו רוצים לכתוב מערך בן 3 איברים מסוג **INT**, נכתוב את הפקודה:

```
fwrite(&numbers,sizeof(int),3,f);
```

כך נכתב כל המערך:

```
int numbers[3];
```

בדוגמא הבאה נכתוב כמה איברים לקובץ:

```
#include <stdio.h>
#include <stdlib.> /* for the exit command */
typedef struct{ /* creating struct */
int number;
}something;
void main()
{
int numbers[10]={1,2,3,4,5,6,7,8,10,12};
int age=15; /* creating array and int */
something abc[4]={
{5},
{10},
{15},
{20}};
FILE *f;

f=fopen("numbers.bin","wb");
/* creating binary file and opening for write. */
if(f==NULL)  /* check if file opened */
{
printf("Error in opening file");
exit(1);
}

/*writing the array */
fwrite(&numbers , sizeof(int), 10, f);

/*writing the int */
fwrite(&n,sizeof(int),1,f);

/*writing the struct*/
fwrite(&abc,sizeof(something), 4, f);

fclose(fp);
}
```

הערה – את תוכן הקובץ לא ניתן לראות בעורך טקסט כי הוא לא קובץ טקסט – אם נפתח אותו נראה תווים משונים.

כעת נעבור על הקוד –

בשני בשורות הראשונות הגדרנו את הספריות בשביל הפקודות.
שורות 3-5 הגדרנו מבנה בשם **something** שיש בו משתנה מסוג **int**.
שורות 6-9 פתחנו את הפונקציה הראשית והגדרנו מערך איברים מסוג **int** ואיבר יחיד מסוג **int**.
שורות 10-14 הגדרנו מערך מסוג **something** – - המבנה שהגדרנו מקודם ואיתחלנו אותו.

שורות 15-22 הגדרנו מבנה **FILE** , פתחנו אותו לכתיבה בינארית ובדקנו כישלון.
החל מ 24 ועד סוף הקובץ כתבנו את המשתנים המערכים והמבנים לקובץ וסגרנו אותו.

הערך המוחזר של **fwrite** - הערך המוחזר של הפונקציה הוא מספר האיברים שנכתבו לקובץ . כדי לבדוק שאין שגיאות בכתיבה – ניתן להשוות ערך זה למספר האיברים שכתבנו:

```
a=fwrite(&something,sizeof(int),22,f);
if(a!=22)
{
printf("ERROR writing to file");
exit(1);
}
```

קריאה מקובץ בינארי:
fread();
המבנה של **fread** הוא בדיוק כמו של **fwrite**
כלומר:

```
fwrite(&abc,sizeof(int),3,f); // כותב לקובץ את המערך
```

```
fread(&abc,sizeof(int),3,f); // קורא מהקובץ למערך
```

וגם, ערכה המוחזר הוא מספר האיברים שנקראו מהקובץ.

בדוגמא הבאה נראה קריאה מקובץ בינארי והדפסת התוצאות.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
FILE *f=fopen("numbers.txt","rb");
int a[3];
int check;
if(f==NULL)
{
printf("error in opening");
exit(1);
}
check=fread(&a,sizeof(int),3,f);
if(check!=3)
{
printf("error in reading");
exit(1);
}
printf("%d %d %d", a[0], a[1], a[2]);
}
```

ובזאת סיימנו את נושא הקבצים הבינאריים.

קבצים – נושאים מתקדמים:

כעת נלמד על מספר נושאים מתקדמים בכל נושא הקבצים:

-מציאת המיקום הנוכחי של מצביע הקובץ.
-שינוי מיקום המצביע.
-זיהוי סוף הקובץ.

מציאת מיקום המצביע:
לכל קובץ קיים מצביע המציין את המיקום הנוכחי של הקובץ.
המיקום הנוכחי הוא המקום של הבית הבא שממנו יש לקרוא או לכתוב לקובץ.
בפתיחה הראשונה – מצביע הקובץ מוצב על תחילת הקובץ.
כל פעולת קריאה או כתיבה תקדם את המצביע.

לעיתים אנו צריכים לדעת את מיקום המצביע:
לשם כך יש את הפונקציה **ftell**.
תבנית הפונקציה:

```
var = ftell(filepointer);
```

var - משתנה אליו יכנס המיקום הנוכחי של הקובץ בביתים.
Filepointer – הקובץ.

ערכה המוחזר של הפונקציה הוא **LONG**, ולכן צריך ליצור משתנה מסוג **LONG**.
דוגמא לתוכנית עם הפונקציה:

```
#include <stdio.h>
void main()
{
FILE *f=fopen("abc.txt","rt");
long s;
s=ftell(f);
printf("%d", s);
}
```

הפלט יהיה כמובן 1 מכיוון שאנו רק פתחנו את הקובץ.

שינוי מיקום המצביע:
הפונקציה **fseek**

בעזרת הפונקציה אפשר לשנות את המיקום **N** בייטים אחורה או קדימה, מנקודת מוצא נוכחית, מההתחלה, ומהסוף.
מבנה:

```
fseek(file,0,SEEK_***);
```

file – קובץ
0 – מספר הצעדים שיש להתקדם – אפשר לכתוב גם מספרים שליליים לדוגמא -1

SEEK_* -** נקודת המוצא:
יש 3 נקודות מוצא – זאת אמרת מאיפה לזוז:

SEEK_SET - תזוזה מתחילת הקובץ
SEEK_CUR - תזוזה מהמיקום הנוכחי
SEEK_END – תזוזה מסוף הקובץ.

הערה: את הנקודות מוצא חייב לכתוב באותיות גדולות!

תוכנית סיכום לשני הפקודות: מציאת גודל קובץ.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
FILE *f=fopen("Textfilesinc.doc","rt");
long size;
if(f==NULL)
exit(1);
fseek(fp,0,SEEK_END); // move to end of file
size=ftell(f); /* get the current "place" of file in
byts.
printf("%d", size);
}
```

התוכנית יודעת את גודל הקובץ ע"י תזוזה לסוף הקובץ והדפסת המיקום הנוכחי בביתים.

מציאת סוף הקובץ – **feof ()**
מבנה הפונקציה:

```
var = feof(FILENAME);
```

var - משתנה
Filename – קובץ

הפונקציה מחזירה אמת (ערך שונה מאפס) כאשר מצביע הקובץ נמצא בסוף, וערך שקר (אפס) כאשר לא הגענו לסוף הקובץ.

וכך סיימנו את כל נושא הקבצים בשפת c.