

files + bitwise

קבצים ופעולות על ביטים

נושא :

I/O files - עבודה עם קבצי טקסט
-פעולות על ביטים

קיימים שני פורמטים לכתיבה/קריאה ל/מקובץ :

- פורמט בינארי
- פורמט טקסט
- בפורמט בינארי הנתונים נשמרים לקובץ בדיוק כפי שהם מאוכסנים בזיכרון. בפורמט טקסט הנתונים מפורקים לתווים והם נשמרים בייצוג התווי(ASCII) שלהם.
- **באופן כללי :** פורמט בינארי הוא חסכוני יותר ואילו פורמט טקסט הוא נח יותר מבחינת היכולת לקרוא ולהבין את הקובץ.
- קיימים מספר שלבים בשמירת/שחזור נתונים אל/מקבצים :
- ❖ פתיחת הקובץ במוד המתאים (קריאה/כתיבה, טקסט/בינארי)
- ❖ ביצוע פעולות הקריאה/כתיבה
- ❖ סגירת הקובץ

files

פתיחה וסגירה של קובץ

■ הפונקציות לפתיחת וסגירת הקובץ הן **fopen** ו-**fclose** בהתאמה

■ הגדרת מצביע לקובץ :

`FILE * pf`

□ פתיחת קובץ :

פתיחת קובץ מתבצעת על ידי קריאה ל-**fopen** המחזירה מצביע ל-**FILE**. אם פתיחת הקובץ נכשלה, אז המצביע מקבל ערך **NULL**. לכן תמיד יש לבדוק את הערך המוחזר מפונקציה זו (הדבר דומה להקצאת זכרון במידה מסויימת).

□ כדי לפתוח את הקובץ "try.txt" לדוגמה, אפשר לבצע זאת :

```
FILE *f = fopen("try.txt" , "rt");
```

```
if(f == NULL)
```

```
{
```

```
    /* Handle case where couldn't open file. */
```

```
}
```

□ הארגומנט הראשון שמקבלת הפונקציה הוא שם הקובץ (אם אינו בתיקית התוכנית, בנתיב מלא או יחסי אליו); הארגומנט השני הוא *מחרוזת סוג הפתיחה*, שעליה נדבר מיד, והיא כאן מבקשת לפתוח לקריאה קובץ טקסט (ברירת מחדל).

files

פתיחה וסגירה של קובץ

■ מחרוזת סוג הפתיחה :

r	read only (need an existed file)
w	write only (delete an existed file)
a	append (to an existed file or a new file)
r+	read and write to an existed file
w+	write and read (delete file if existed)
a+	append and read file – don't use

■ סגירת קובץ

אחר השימוש בקובץ יש לסגור אותו. הפונקציה שמבצעת זאת היא `fclose` המקבלת כארגומנט מצביע לקובץ. משתמשים בה בצורה הבאה :

```
FILE * pointer_to_file;
```

```
pointer_to_file = (fopen.....  
.....
```

```
fclose (pointer_to_file);
```

חשוב! פקודה `exit()` שמסימת את ריצת התכנית, לפני הסיום סוגרת את כל הקבצים הפתוחים! אין צורך להשתמש ב-`fclose()` לפני `exit()` !!

חיפוש בקבצים (הזזת מצביע)

הפונקציה `fseek` מאפשרת להזיז את הסמן הקריאה/כתיבה של הקובץ ממקומו הנוכחי למיקום מסויים בקובץ :

`void fseek (FILE* fp, long offset, int origin)`

הפונקציה מקבלת 3 פרמטרים :

- ❖ `fp` - מצביע לקובץ
- ❖ `offset` - מספר הבתים להזזה ביחס ל-`origin`
- ❖ `origin` - מיקום התחלתי. ערך זה יכול להיות אחד משלושה :
 - `SEEK_SET` - התחלת הקובץ
 - `SEEK_CUR` - המיקום הנוכחי של סמן הקריאה/כתיבה של הקובץ
 - `SEEK_END` - סוף הקובץ

תו סיום הקובץ – EOF

ישנו תו מיוחד, EOF, המסמן לפי מוסכמה את סיום התוכן האמיתי של קובץ. ערך המספרי שלו הוא -1. אין צורך "לכתוב" אותו לקובץ, אלא רק להשתמש. לפי המוסכמה, סוף הקובץ – זהו בית הראשון מחוץ לקובץ. תמיד ניתן להניח שבכל קובץ אפשר לבדוק האם הגענו לסוף הקובץ או לא.

files

קריאה וכתיבה לקובץ טקסט

- פונקציה לקריאה מפורמטת מקובץ
- **int fscanf(FILE* stream, const char *format[,address,...])**
- פונקציה לכתיבה מפורמטת לקובץ
- **int fprintf(FILE* stream, const char *format[,argument,...])**
- פונקציה לקריאת מחרוזת מקובץ
- **char * fgets (char* s, int n, FILE*stream)**
- פונקציה לכתיבת מחרוזת לקובץ
- **int fputs (const char* s, FILE*stream)**
- פונקציה לקריאת תו בודד מקובץ
- **int fgetc (FILE* stream)**
- פונקציה לכתיבת תו בודד לקובץ
- **int fputc (int c, FILE* stream)**
- הערה חשובה: כל פקודת קלט/פלט מקדמת מצביע!
- קלט/פלט סטנדרטי:
 - **stdin** - standard input file (keyboard).
 - **stdout** -standard output file (screen).
 - **stderr** - standard error file (screen)

files

דוגמא 1 (קבצי טקסט)

שימוש בפקודה fprintf

```
#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;
}
```

files

דוגמא 2(קבצי טקסט)

שימוש בערך המוחזר מ-`fscanf`. כמו פונקציה `scanf`, גם `fscanf` מחזירה את כמות הנתונים שהיא קלטה. הפונקציה מחזירה `EOF(-1)` בהגעה לסוף הקובץ.

```
#include <stdio.h>

#define MAX_SIZE 10

int main()
{
    FILE *fin;
    int iCounter = 0;
    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", &numbers[iCounter]) > 0)
    {
        printf("%d\n", numbers[iCounter++]);
    }

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

files

דוגמא 3 (קבצי טקסט)

שימוש בפקודה fgets. הפקודה מחזירה NULL בהגעה לסוף הקובץ.

```
#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;
}
```


קבצים - המשך

פקודות מתוך stdlib.h :

atoi, atof, atol, itoa, ltoa,.....

atoi – הפיכת מחרוזת למספר מסוג int

int atoi(const char *)

on success : returns an integer number,

on error : returns 0.

דוגמא:

```
char *str = NULL;
```

```
int value = 0;
```

```
/* An example of the atoi function. */
```

```
str = " -2309 ";
```

```
value = atoi( str );
```

```
printf( "Function: atoi( \"%s\" ) = %d\n", str, value );
```

```
/* Another example of the atoi function. */
```

```
str = "3a412";
```

```
value = atoi( str );
```

```
printf( "Function: atoi( \"%s\" ) = %d\n", str, value );
```

Output:

Function: atoi(" -2309 ") = -2309

Function: atoi("3a412") = 3

files

ftell- ו fgets

שימוש נוסף בפקודה fgets:

נתון קובץ טקסט שמכיל רצף של ספרות ללא שום תו מפריד ביניהם. ידוע שכל 3 בתים מהווים מספר. יש לקלוט את המספר הראשון מהקובץ למשתנה מסוג int. מצביע f נמצא בתחילת הקובץ.

```
int num;  
char str[4];  
fgets(str,4,f);  
num = atoi(str);
```

קליטת 3 תוים למחרוזת בגודל 4. הפקודה יודעת לבנות רק מחרוזות

הפיכת המחרוזת למספר

פקודה ftell:

פקודה long ftell(FILE*stream) מחזירה מספר הבתים מתחילת הקובץ עד המצביע.

דוגמא לשימוש בפקודה:

נתון קובץ טקסט שמכיל רצף של ספרות ללא שום תו מפריד ביניהם. גודל הקובץ וכמות המספרים בו לא ידועים. ידוע שכל 3 בתים מהווים מספר. יש לחשב את כמות המספרים בקובץ. מצביע f נמצא בתחילת הקובץ.

```
int size;  
fseek(f, 0, SEEK_END);  
size = ftell(f)/3;  
fseek(f,0,SEEK_SET);
```

העברת מצביע לסוף הקובץ

החזרת מצביע לתחילת הקובץ

files

דוגמא 4 - fseek ו- ftell

נתון קטע קוד הבא :

```
#include <stdio.h>

int main( )
{
    .....
    fseek (ifp, -1, SEEK_END);
    while ( ftell (ifp) >= 0 )
    {
        c = fgetc(ifp);    /*read one character from file*/
        putchar(c);        /*print it to the screen*/
        fseek(ifp, -2, SEEK_CUR);
    }
    .....
}
```

מה מבצע קטע קוד הנ"ל?

קבצים - המשך

פקודות מתוך stdio.h :

- **rewind** – החזרת מצביע לתחילת הקובץ

`void rewind(FILE* stream)`

הפקודה `rewind(stream)` שקולה ל-
`.fseek(stream,0,SEEK_SET)`

- **remove** - מחיקת קובץ.

`int remove(const char *filename)`

on success : returns 0,

on error : returns -1.

- **rename** - שינוי שם הקובץ

`int rename(const char *oldname, const char newname)`

on success : returns 0,

on error : returns -1.

- **feof** – בדיקה האם הגענו לסוף הקובץ

`int feof(FILE*stream)`

on success : returns non- zero ,if an end-of-file
indicator was detected

on error : returns 0.

files

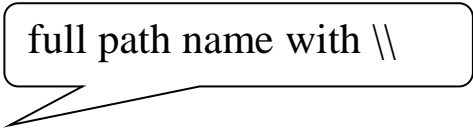
דוגמא 5 (קבצי טקסט)

בדוגמא הזאת קולטים רשימה של משפטים מהמשתמש וכותבים אותם לתוך קובץ – כל משפט בשורה נפרדת. אחרי זה קולטים את המשפטים מהקובץ ומדפיסים למסך השחור.

```
#include<stdio.h>
#include<stdlib.h>
#define STRING 100

int main( )
{
    char path[ ] = "d:\\output.txt";
    char input[STRING] ;
    FILE *iofp;

    /*create new file*/
    if ((iofp = fopen(path, "w+t")) == NULL)
    {
        printf("Cannot open file...\n");
        printf("Sorry...but that is all for now!\n");
        exit(1);
    }
```



דוגמא 5- המשך

*/*Using fgets for input from keyboard.*

*The newline character is included in the string */*

do

■ קריאה מהמקלדת

{

printf("Enter a string (to exit - only <Enter>): \n");

fgets(input ,100,stdin);

fputs(input, iofp); */*with new line*/*

} while(*input != '\n');

fseek(iofp,0,SEEK_SET);

while(fscanf(iofp, "%s", input) >0)

puts(input);

fclose(iofp);

return 0 ;

}

files

דוגמא 6 (קבצי טקסט)

דוגמא לקריאת תוכן מקובץ קלט והדפסתו למסך השחור.

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

int main( )
{
    /*read from file , display data*/
    FILE *fin;
    int ch; // must b int!!
    fin = fopen("textfile.txt", "rt");
    /*open text file */
    if (fin==NULL)
    {
        printf("Unable to open file...\n");
        exit(1); /* for error */
    }
    do {
        ch = fgetc(fin); /*read one char*/
        putchar(ch); /* display char*/
    } while (ch!=EOF);
    fclose(fin); /*close file*/
    return 0;
}
```

bitwise

פעולות על ביטים – סקירה כללית

אופרטורים הפועלים על סיביות מתייחסים לצורה בה אנחנו מייצגים נתונים במחשב.

בעוד שפעולות הרגילות בשפת C, כגון חיבור, חיסור וכו' מתייחסים אל המספר עצמו, הפעולות הבינאריות מתייחסות אל הסיביות השונות המרכיבות את המספר.

אפשר לבצע פעולות על ביטים עבור כל המספרים שלמים (משתנים או קבועים).

אין לבצע פעולות על ביטים עבור משתנים מסוג float, double, long double

תוצאה של הפעולות האלו תמיד מספר שלם.

bitwise

פעולות על ביטים שקיימות הן :

&	AND
 	OR
^	XOR
~	NOT
<<	Shift Left
>>	Shift Right

Bitwise Operators – true table

a	b	a&b	a b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

bitwise

- פעולת AND מעתיקה 1 לתוצאה אם 1 מופיע בשני האופרנדים.

```
int main()
{
    unsigned int a = 60;    /* 60 = ...0011 1100 */
    unsigned int b = 13;    /* 13 = ... 0000 1101 */
    unsigned int c = 0;
    c = a & b;               /* 12 = ...0000 1100 */
}
```

■ התוצאה תמיד בביט 10, אפילו שהפעולה עצמה מתבצעת בביט 2

- פעולת OR מעתיקה 1 לתוצאה אם 1 מופיע באופרנד אחד לפחות.

```
int main()
{
    unsigned int a = 60;    /* 60 = ...0011 1100 */
    unsigned int b = 13;    /* 13 = ...0000 1101 */
    unsigned int c = 0;
    c = a | b;               /* 61 = ...0011 1101 */
}
```

bitwise

- פעולת **XOR** מעתיקה 1 אם 1 מופיע באופרנד אחד בלבד (ולא בשניהם).

```
main() {  
    unsigned int a = 60;    /* 60 = 0011 1100 */  
    unsigned int b = 13;    /* 13 = 0000 1101 */  
    unsigned int c = 0;  
    c = a ^ b;              /* 49 = 0011 0001 */  
}
```

- פעולות **Shift Left, Shift Right**

```
main() {  
    unsigned int Value=4;    /* 4 = 0000 0100 */  
  
    Value = Value << 2;      /* 16 = 0001 0000 */  
    printf("%d\n", Value);   /* Prints 16 */  
}
```

bitwise

דוגמא 1

אלגוריתם ההחלפה של XOR.

בתכנות, אלגוריתם ההחלפה של XOR הוא אלגוריתם אשר משתמש בפעולה לוגית XOR בכדי להחליף ערכים של שני משתנים, בעלי אותו סוג מידע, וללא שימוש במשתנה עזר זמני.

```
#include <stdio.h>
main()
{
    int One = 20, Two = 12;

    printf("One = %d Two = %d\n", One, Two);
    One ^= Two;
    Two ^= One;
    One ^= Two;
    printf("One = %d Two = %d\n", One, Two);
}
```

קוד זה ניתן לכתוב גם בצורה של ביטוי אחד, אשר משתמש בתכונה של האסוציאטיביות מימין של ההשמה ב-C :

$x^{\wedge}=y^{\wedge}=x^{\wedge}=y;$

bitwise

דוגמא 1 - המשך

■ לדוגמא, נניח שיש לנו שני ערכים $X = 12$ ו- $Y = 10$.

■ בבינארית זה יראה כך:

$$X = 1\ 1\ 0\ 0$$

$$Y = 1\ 0\ 1\ 0$$

■ כשנפעיל XOR על שני המשתנים אנו נקבל 0 1 1 0, אותו אני נעתיק למשתנה X. עתה יש לנו:

$$X = 0\ 1\ 1\ 0$$

$$Y = 1\ 0\ 1\ 0$$

■ כשנפעיל XOR שוב, נקבל 1 1 0 0. נעתיקו למשתנה Y. עתה יש לנו:

$$X = 0\ 1\ 1\ 0$$

$$Y = 1\ 1\ 0\ 0$$

■ לבסוף, נפעיל XOR ונקבל 1 0 1 0 - אותו שוב נשים ב X. ■ קיבלנו:

$$X = 1\ 0\ 1\ 0$$

$$Y = 1\ 1\ 0\ 0$$

■ כלומר, המשתנים החליפו את הערכים ביניהם.

bitwise דוגמאות ל-mask-

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

```
int GetBit(unsigned int number, int bit);  
unsigned int SetBit(unsigned int number , int bit);  
unsigned int ResetBit(unsigned int number , int bit);  
unsigned int TwoPower(int);
```

```
int main()  
{  
    unsigned int res, x = 33, bit;  
    res = TwoPower(5);  
    bit = GetBit(x,5);  
    x = SetBit(x,8);  
    x = ResetBit(x,8);  
    return 0;  
}
```

bitwise

דוגמאות ל-mask

/*Q: I want to raise a number to a power of $x 2^p$?

A :*/

```
unsigned int TwoPower(int p)
{
    unsigned int unity = 1;

    return unity<<p;
}
```

/*Q: I want to get a bit of position bit from number?

A :*/

```
int GetBit(unsigned int number , int bit)
{
    unsigned int unity = 1;
    res = number & (unity << bit);
    return res!=0;
}
```

מסכה שבעזרתה בודקים את ערכו של
ביט מסויים

bitwise

דוגמאות ל-mask

/*Q: I want to set a bit of position bit from number by 1?

A : */

```
unsigned int SetBit(unsigned int number , int bit)
```

```
{
```

```
    unsigned int unity = 1;
```

מסכה שבעזרתה משנים את ערכו של
הביט מסויים ל-1

```
    return number | (unity << bit);
```

```
}
```

/*Q: I want to reset a bit of position bit from number by 0?

A : */

```
unsigned int ResetBit(unsigned int number , int bit)
```

```
{
```

```
    unsigned int unity = 1;
```

```
    return number & ~(unity << bit);
```

```
}
```

מסכה שבעזרתה משנים את ערכו של הביט מסויים ל-0