C programming Language

Chapter 6:

F i l e s

זיכרון המחשב

מהירות ומחיר

גודל

אוגרים – register
זמן גישה: מיידי
כמות: מצומצת

RAM– זיכרון ראשי
זמן גישה: 5-70 ננו-שניה
כמות: O(GB)

דיסק – זיכרון מישני
זמן גישה: 2-8 מילי-שניה
כמות: O(TB)

קבצים יושבים פה

קבצים – מקורות מידע במחשב

- ישנם שני סוגי קבצים:
  - קובץ טסקט (text file):
    - כל מספר/תו מיוצג ע"י ערך ה- ascii שלו.
    - ניתן לקריאה ע"י עורך (editor) כמו: vi, notepad, emacs.
  - קובץ בינארי (binary file):
    - מיועד להחזקת נתונים, ולא ניתן לקריאה ע"י עורך.
    - כל מספר/תו תופסים בקובץ בדיוק אותה כמות של בתים כפי שהם תופסים בזיכרון. למשל, המספר 1234.56789 יתפוס בקובץ טקסט 10 בתים ואילו בקובץ בינארי 4 בתים (float).
    - מיועד לשמירה וגישה מהירה לכמות גדולה של נתונים.

קבצים

- במהלך תוכנית נרצה לפתוח קבצים לקריאה / כתיבה.
- ישנן מספר פונקציות המאפשרות עבודה עם קבצים. כולן נמצאות בספריה:

stdio.h

- על מנת לעבוד עם קבצים נעביר אותם לזיכרון מהיר יותר (נקרא אותם ל-RAM).
- התהליך הזה נקרא "פתיחת קובץ".
- בעת פתיחת הקובץ אנו מקבלים "מצביע לקובץ"

FILE*

מהו *FILE ?

- לא מעניין אותנו!
- אבל:
- *FILE הוא מבנה המכיל כל מיני נתונים על הקובץ שפתחנו ומאפשרים למערכת לעבוד איתו.
- אנו עובדים על **מצביע** לקובץ כי בחלק גדול מהפונקציות שנעבוד איתן אנו **נשנה** את הקובץ! ולכן אנו ושלחים אותו by address!

פונקציות עבודה עם קבצים

- כמו בקלט ופלט הסטנדרטיים גם בקלט ופלט מקבצים אנו נעבוד עם פונקציות שכבר בנויות!
- ספריית stdio.h.
- בעבודה עם קבצים צריך:
  1. לפתוח את הקובץ (העתקה לזיכרון יותר מהיר)
  2. לקרוא / לכתוב
  3. לסגור את הקובץ (שחרור משאבים והעתקה לזיכרון האיטי).
- לכל שלב יש לנו פונקציות מתאימות שמאפשרות לנו לבצע את הפקודות הנ"ל.

Open File

FILE* fopen(const char* filename , const char* mode)

כאשר :

filename :

הינו מצביע למחרוזת תווים שהינה שם הקובץ אותו רוצים לפתוח

(יש לציין מסלול מלא ביחס למיקום התוכנית) .

- mode :

הינו מצביע למחרוזת תווים אשר מציינת את מטרת פתיחת הקובץ . האופציות הן :

"r" - פתיחת קובץ טקסט לקריאה.

"w" - פתיחת קובץ טקסט לכתיבה.

Open File (cont.)

FILE *fopen (char **fileName* , *char* **mode);*

mode – *המשך:*

"a" – פתיחת קובץ לכתיבה בסופו (append)

אם הקובץ לא קיים – נכשלה הפתיחה

"r+" – פתיחת קובץ לקריאה/לכתיבה

אם הקובץ לא קיים נכשלה הפתיחה

"w+" – פתיחת קובץ חדש לקריאה/לכתיבה

אם הקובץ קיים – הישן נמחק וכותבים מראשיתו

"a+" – פתיחת קובץ לקריאה/הוספה לסוף הקובץ

אם הקובץ לא קיים – נוצר קובץ חדש

Open File (cont.)

- במידה והפונקציה fopen() הצליחה בפתיחת הקובץ יוחזר מצביע לקובץ, אחרת יוחזר NULL .

- מספר נקודות :
  - 1)במידה ונפתח לכתיבה או הוספה, קובץ שלא קיים, יווצר קובץ בשם זה.
  - 2)במידה ונפתח לכתיבה קובץ שקיים, הכתיבה לקובץ זה תמחק את הקיים (overwrite).
  - 3)במידה ונפתח לקריאה קובץ שלא קיים או ללא הרשאה מתאימה, הפונקציה תחזיר NULL.
- כפי שניתן לראות הפונקציה fopen() מחזירה מצביע לקובץ שאיתו אנו רוצים לעבוד. בכדי לשמור כתובת זו נגדיר משתנה מטיפוס FILE* .
- אם הפונקציה תחזיר למשתנה זה NULL סימן שהפתיחה לא הצליחה ולכן אין טעם להמשיך במהלך ביצוע התוכנית.

פתיחה של קבצים בינאריים

- במקרה של קבצים בינאריים מדובר באותן פונקציות ומודים (mode), רק שיש צורך ליידע את המחשב שמדובר בקובץ בינאריים.
- בקובץ בינארי הקריאה והכתיבה היא לא של bytes אלא לפי טיפוסים.
- דבר זה נעשה על ידי הוספת התו 'b' ל-string המגדיר את ה-mode של הקובץ. למשל:
  - "rb" יפתח קובץ בינארי לקריאה
  - "a+b" שקול ל "a+" לקובץ בינארי.

Example

לדוגמא, פתיחת הקובץ "in.dat" לקריאה :

```
 FILE*  fin ;
fin=fopen("in.dat" , "r");
if( fin==NULL ) {
       printf("Error in opening file %s\n", "in.dat") ;
        exit(1) ;
}
```

הסבר :

1) ראשית, הצהרנו על fin כמצביע לקובץ.

2) קריאה ל - fopen() עם שם הקובץ אותו אנו רוצים לפתוח באופן פתיחה "r" , כלומר לקריאה.

3) לאחר הקריאה לפונקציה fopen() בדקנו אם הקריאה נכשלה, אם כן מודפסת הודעה מתאימה ומתבצעת קריאה לפונקציה exit() . כאשר נהוג להחזיר 1 כמציין סיום לא נורמלי של התוכנית.

Example


```
#include<stdio.h>
int main() {
        FILE *ifp, *ofp;
        char *mode = "r";
        char outputFilename[] = "out.list";
        ifp = fopen("in.list", mode);
        if (ifp == NULL) {
                fprintf(stderr, "Can't open input file in.li
                exit(1);
        }
        ofp = fopen(outputFilename, "w");
        if (ofp == NULL) {
                fprintf(stderr, "Can't open output file %s!\n",
outputFilename);
                exit(1);
        }
        … //We must close the file
}
```

שימו לב כי הקובץ שאנו רוצים לקרוא חייב להיות קיים כבר!

אם קובץ הכתיבה קיים כבר הוא ימחק, אחרת ייווצר חדש.

סגירת קובץ

- בסיום השימוש בקובץ או עם סיום התוכנית חייבים לסגור את הקבצים שפתחנו. אם לא נעשה זאת, חלק מהמידע שכתבנו לקובץ עלול להיאבד. לצורך כך קיימת פונקצית הספרייה fclose() אשר אב הטיפוס שלה מוגדר גם כן ב - stdio.h והוא :

```
int  fclose( FILE* )
```

פונקציות לעבודה עם קבצים

- רוב שמות הפונקציות אותם אתם מכירים לקלט/פלט.
- לשם הפונקציה יתווסף בדר"כ f בהתחלה, וכן ישלח המצביע לקובץ. למשל:
  - int scanf(const char* *format*, ...);
    int fscanf(FILE* *stream*, const char* *format*, ...);
  - int printf(const char* *format*, ...);
    int fprintf(FILE* *stream*, const char* *format*, ...);
  - char* gets(*char* s*);
    char* fgets(*char* s*, *int n*, *FILE* stream*);

קריאה מקובץ

- נניח שהקובץ קלט שלנו מכיל בכל שורה שם וציון:

```
in.list
-------
foo 70
bar 98
...
```

- אנו נשתמש בקובץ שפתחנו (קודם) –ונעתיק את המידע לקובץ הפלט, תוך תוספת פקטור של 10 נקודות לכל אדם.

- איך נדע שהגיע סוף הקובץ?

סוף קובץ - EOF

- הפונקציה fscanf (כמו scanf) מחזירה ערך! הערך שמוחזר הוא מספר הערכים שהפונקציה קראה בהצלחה.
- אך כאשר הפונקציה מגלה שהיא הגיעה לסוף הקובץ היא מחזירה ערך מיוחד: EOF (End Of File).

```
char username[.rahc llun rof artxe enO // ;[9
int score;
 ...
while(fscanf(ifp, "%s %d", username, &score) != EOF) {
        fprintf(ofp, "%s %d\n", username, score+10);
}
…
```

סוף קובץ – EOF (2)

- הבעיה היא אם יש קלט לא תקין (מבנה הקובץ הוא לא כפי שהובטח) ו- fscanf יכולה להיתקע ואז:
  - או שניתקע בלולאה אינסופית.
  - או שלא נגלה האם זה בגלל קלט לא תקין או בגלל הגעה לסוף הקובץ.
- לכן הרבה מעדיפים להשתמש בפונקציה feof(FILE*).
  - הפונקציה מקבלת מצביע לקובץ ומחזירה האם הגענו לסוף הקובץ או לא בלי קשר להצלחת הקריאה.

```
while (!feof(ifp)) {
        if (fscanf(ifp, "%s %d", username, &score) != 2) break;
        fprintf(ofp, "%s %d", username, score+10);
}
```

סוף קובץ – EOF (2)

- הבעיה היא אם יש בלוק לא תקין (מרוב הקורס הוא לא כפי שהו
  - או
  - או
- לכן
  - הס
  בל

```
while (
    if (fscanf(ifp, "%s %d", username, ascore) != 2) break;
    fprintf(ofp, "%s %d", username, score+10);
}
```

שימו לב כי גם EOF וגם הפונקציה feof() יזהו את סוף הקובץ רק לאחר שינסו לקרוא **מעבר** לסוף הקובץ. זאת אומרת **אחרי** הקריאה התקפה האחרונה.

פונקציות קלט/פלט של תו בודד

- קריאת תו:

char fgetc (FILE* *filePointer);*

מחזיר את התו הבא בקובץ המוצבע ע"י filePointer. לאחר פעולת הקריאה, הפונקציה מקדמת את המצביע. סוף הקובץ מסומן ע"י הערך – EOF

- כתיבת תו:

int fputc (char *ch , FILE * filePointer);*

כותב את התו שב-ch לתוך קובץ המוצבע ע"י filePointer. לאחר פעולת הכתיבה, הפונקציה מקדמת את המצביע.

דוג' להעתקת קבצים


```
#include <stdio.h>
int main( int argc, char* argv[] ) {
   FILE *src, *trg;
   int c;

   if (argc != 3) {  printf("Bad usage \n");   return 1;  }
   src = fopen(argv[1], "r");
   trg = fopen(argv[2], "w");
   if(NULL == src || NULL == trg) {
         printf("Failed opening files");
         return 1;
   }
   while (EOF != (c=fgetc(src))) {  // fgetc – read char from file
         fputc(c, trg);                    // fputc – write char to file. One
   should check if succ.
   }
   fclose(src); fclose(trg);
}
```

פונקציות קלט/פלט של רצף תווים

- קריאת רצף תווים:

char fgets(char* line, int max, FILE* stream);*

קורא לכל היותר את max-1 התווים (שומר מקום ל-'\0') מהקובץ המוצבע ע"י stream לתוך line. יעצור אם נתקל ב-'\n' (לאחר שהעתיקו לתוך line). יחזיר את line, או EOF.

- כתיבת רצף תווים:

char fputs(const char* line, FILE* stream);*

כותב את רצף התווים שב-line, לתוך קובץ המוצבע ע"י stream. לא יכתוב את '\0' שבסוף line לתוך הקובץ.

Example

- Write program that uses **fgets** to read lines from the input file. Then the program uses **fputs** to put words into the output file retaining one space between the words.
- The maximal line size is 80 characters.

Solution


```
#include <stdio.h>
#include <string.h>
#define MAX 80
 void main(){
   FILE *source, *result;
   char string [MAX+1];
   char *save, *run;
   int i=0, mode=1;
   if((source=
         fopen("source.txt","r"))==NULL){
             printf("Can't open the
   file\n");
             exit(1);
   }
   if((result=
         fopen("result.txt","w"))==NULL)
   {
         printf("Can't open the file\n");
         exit(1);
   }
```

```
while(fgets(string,MAX+1,source)){
     save = run = string;
     while(*run){
        if(mode) {
             if(*run==' ')
                mode=0;
             *save++ = *run;
        }
        else
             if(*run!=' ') {
                mode=1;
                *save++ = *run;
             }
        run++;
     }
     *save='\0';
     fputs(string,result);
   }
   fclose(source);
   fclose(result);
}
```

Formatted Input/Output

- int fscanf(FILE* *stream*, const char* *format*
  [, argument]... **);**

- int fprintf(FILE* *stream*, const char* *format*
  *[, argument]...***);**

Example

- Write program that uses **fscanf** and **fprintf** to read words from the input file and places them into the output file with one space a separator.
- The maximal word size is 80 characters.

Solution


```
#include <stdio.h>
#include <stdlib.h>
#define MAX 80
 void main(){
   FILE *source, *result;
   char string [MAX+1];
   if(((source=fopen("source.txt","r"))==NULL) ||
      ((result=fopen("result.txt","w"))==NULL)) {
        printf("Can't open the file\n");
        exit(1);
   }
   while((fscanf(source,"%s", string))!=EOF)
        fprintf(result,"%s ",string);
fclose(source);
fclose(result);
}
```

fscanf - format specification Fields

- A format specification has the following form:

  %[*] [width] type

- If the first character in the set is a caret (^), the effect is reversed: The input field is read up to the first character that does appear in the rest of the character set.
- For example:

```
fscanf( ptr, "%20[^#\n]%9d%*c", name, &id);
```

  - the function fscanf reads 20 characters, or till letter ('#') , or till newline from the input stream and stores them in field *name,* then it reads the next 9 characters and converts them into integer *id,* then it reads one symbol which is not stored.

תזוזות בקובץ

- איך אנו זזים בקובץ?
  - כל קריאה ממשיכה מאיפה שסיימנו לקרוא.
  - על כתיבה ממשיכה מאיפה שכתבנו בעבר.
  - אפשר לקרוא ולכתוב לאותו קובץ במקומות שונים!
  - append מול trunk!
- אינדקסים בקובץ!
- יש לנו אנדקס למיקום קריאה ואינדקס למיקום כתיבה.
- כל קריאה / כתיבה מזיזה את האינדקס הרלוונטי.
- יש לנו אפשרות להזיז את האינדקס בעצמנו.

גישה אקראית לקובץ – הזזת האינדקסים

קפיצה למיקום בקובץ:

int **fseek (FILE *fp , long offset , int fromWhere);**

- fp – מצביע לקובץ.
- offset – כמות הבתים שרוצים לדלג (היסט חיובי / שלילי)
- fromWhere – מהיכן למדוד את ההיסט:

0 – SEEK_SET – ההיסט ימדד מתחילת הקובץ

1 – SEEK_CUR – ההיסט ימדד מהמיקום הנוכחי

2 – SEEK_END – ההיסט ימדד מסוף הקובץ

ערך מוחזר – הצלחה: 0, כישלון: 0! (לא אפס).

גישה אקראית לקובץ – קבלת מיקום

- קבלת המיקום בקובץ:

long **ftell(FILE *fp);**

  - fp – מצביע לקובץ
  - ערך מוחזר – הצלחה: מיקום המצביע בקובץ, כישלון: -1L

- קפיצה לראש הקובץ:

void **rewind(FILE* fp);**

שקול ל:

fseek(*stream*, 0L, SEEK_SET);

סגירת קובץ

fclose (FILE* *filePointer);*

- אם פתחנו קובץ ואין בו יותר שימוש, חשוב לסגרו.
  - משאבים תפוסים שלא משתחררים מעצמם...
  - buffered output files.
- בתום התוכנית יש לסגור את כל הקבצים אותם פתחנו.

דוג' – קריאת קובץ הפוך


```
int main ( int argc, char* argv[] ) {
  FILE *fp;
  if (argc != 2) { printf("Bad usage \n");   return 1;  }
  fp = fopen(argv[1], "rb");
  if(NULL == fp) { printf("Failed opening file"); return
  1; }
  fseek(fp, -1L, SEEK_END);// move to the last char of the
  file
  do {
       int c = fgetc(fp);   // move ahead one character
       putchar(c);
  }  while (0==fseek(fp, -2L, SEEK_CUR)); // back up 2
  chars
   fclose(fp);
}
```

קבצים ו-buffers

- פעולה של קריאה/כתיבה מדיסק היא פעולה יקרה (איטית).

לכן, הפונקציות משתמשות ב- buffer (מאגר).

- בעת קריאת תו, המערכת תקרא כבר מאגר שלם (buffer).
- באותו אופן, לפני כתיבה לקובץ המערכת תכתוב ל-buffer.
- בהתמלא המאגר, נכתוב זאת לקובץ.
- פעולה זאת היא שקופה למתכנת (מתבצעת מאחורי הקלעים).
- זאת אחת הסיבות שחייבים להקפיד על סגירת קבצי כתיבה!

באם רצונכם לנקות (להכריח כתיבה) את מאגר:

```
int fflush(FILE* fp);
```

fflush(NULL): יבצע פעולת flush על כל הפלטים

כתיבת בלוק נתונים

int **fwrite (const void *buffer , int size , int amount , FILE *fp);**

- כתיבת מקסימום amount איברים, כל אחד בגודל size, מ-buffer לקובץ המוצבע ע"י fp.
- בסיום, fp מקודם כמספר הבתים שנכתבו בפועל.
- ערך מוחזר: הצלחה: מספר האיברים (לא הבתים) שנכתבו.

  כשלון: מספר הקטן מ-amount

קריאת בלוק נתונים

int **fread (const void *buffer , int size , int amount , FILE *fp);**

- קריאת מקסימום amount איברים, כל אחד בגודל size, מהקובץ המוצבע ע"י fp לתוך ה-buffer.
- בסיום, fp מקודם כמספר הבתים שנכתבו בפועל.
- ערך מוחזר: הצלחה: מספר האיברים (לא הבתים) שנכתבו.

  כשלון: מספר הקטן מ-amount

דוג' – קובץ בינארי


```c
typedef struct {
   int     x;
   double  y;
} Nums;

int main( int argc, char* argv[] ) {
   FILE *src=NULL;
   Nums data[] = {{1, 1.1}, {2, 2.2}, {3, 3.3}}, *trg;
   int i, num, SIZE = sizeof(data)/sizeof(Nums);

   if (argc != 2) {printf("Bad usage \n");   return 1;
   }

   // write the binary data
   src = fopen(argv[1], "wb");
   if(NULL == src) { printf("Failed opening file %s
   for writting", argv[1]); return 1; }

   num = fwrite(data , sizeof(Nums) , SIZE , src);
   printf("Wrote %d items to %s.\n", num, argv[1]);
   fclose(src);
```

```c
   // read the binary data
   src = fopen(argv[1], "rb");
   if(NULL == src) { printf("Failed opening
   file %s for reading", argv[1]); return 1;
   }

   trg = (Nums *)
         malloc (SIZE * sizeof(Nums));
   if(NULL==trg) {
       printf("Failed mallocing \n");
   return 1; }
   num = fread(trg, sizeof(Nums) ,
                                        SIZE
   , src);
   printf("Read %d items\n" , num);

   // print the read data
   for(i=0; i<SIZE; i++)
      printf("trg[%i]= <%d, %lf> \n",
                     i, trg[i].x, trg[i].y);

   fclose(src);
}
```