סיכום מבחן 2 – C

Struct:

ביטויים בstruct

```
struct test {
    int num;
    int *pnum;
};

struct test a;
struct test *b;
```

a.pnum	כתובת של	pnum
*a.pnum	הערך שיש בתוך	pnum
(*a).pnum	error	
b.pnum	error	
*b.pnum	error	
(*b).pnum	כתובת של	pnum

קלט של מחרוזת לstruct

תרגיל

- כתבו קטע קוד, המקבל מחרוזת תווים s בצורה שתוארה שלעיל ומציב את פרטי החשבון בתוך מבנה a מהסוג שהגדרנו.

```
sscanf(s,"%u %s %d", &a.num, a.name, &a.balance);
```

```
typedef struct{
    unsigned int num1;
    char name[15];
    double balance;
} acount;
```

Sizeof בStruct: הגודל של הstruct הבא יהיה 32 כי במבנה שלו עושים padding למשתנה האחרון double כי הוא הטיפוס הגדול ביותר.(אין תלות במספר הcharים שהוזנו השטח שמוקצה להם הוא בכל מקרה 15 בתים.

Union:

```
union foo {
   int num;
   char str1[6];
};


int main()
{
   union foo x;
   x.num = 4095;
   printf ("%d\n", x.num);
   strcpy(x.str1,"Hi!");
   printf ("%s\n", x.str1);
   return 0;
}
```

הפלט: 4095

!Hi

Sizeof בUnion: הגודל של הUNion יהיה כגודל המשתנה הגדול יותר שמרכיב אותו.

לדוגמא במקרה מימין הגודל יהיה 6.

:argc, argv

פרמטרים לתוכנית ראשית

- בשפת C ניתן לכתוב את התכנית הראשית בשני אופנים:

```
int main()
```

```
int main(int argc, char* argv[])
```

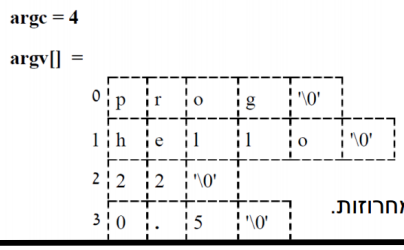
- האפשרות השנייה מאפשרת להעביר פרמטרים מסביבת העבודה אל התכנית, דרך שורת הפקודה (in line arguments)
- הפרמטר argc הוא מספר שלם המונה כמה פרמטרים הועברו לתכנית.
- הפרמטר argv הוא מערך של מחרוזות השומר את הפרמטרים שהועברו.
- תמיד הפרמטר הראשון (המחרוזת הראשונה המאוחסנת ב-argv) היא שם התכנית.

פרמטרים לתוכנית ראשית

- לדוגמא, אם שם התכנית הוא prog.c והיא הודרה לקובץ ביצוע בשם prog.exe הפעלתה עם הפרמטרים הבאים

```
C:\Users\Administrator>prog hello 22 0.5
```

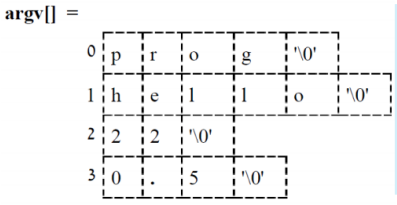
- תגרום לכך ש- argc ו- argv ייראו כך:



- כלומר, הפרמטרים מועברים לתכנית כמחרוזות.

פרמטרים לתוכנית ראשית – דוגמא 1

- אם התכנית מעונינת לשלוף את המספרים מהמחרוזות היא יכולה לבצע זאת ע"י sscanf. לדוגמא:




```
int main(int argc, char* argv[])
{
    int num;
    float fnum;
    sscanf(argv[2], "%d", &num);
    sscanf(argv[3], "%f", &fnum);
    printf("Input parameter %s %f", argv[1], fnum);
}
```

```
C:\Users\Administrator>prog hello 22 0.5
Input parameter hello 0.500000
```

פרמטרים לתוכנית ראשית – דוגמא 2

- תכנית המקבלת כפרמטר שם של קובץ, ויוצרת עבורו גיבוי:

```
#include <stdio.h>
#include <string.h>
int main(int argc, char* argv[])
{
    unsigned char byte;
    FILE* original, *backup;
    if (argc != 2)
    {
        printf("Incompatible number of parameters!");
        return 1;
    }
    original = fopen(argv[1], "rb");
    backup = fopen(strcat(argv[1], "2"), "wb");
    if (original == NULL || backup == NULL)
    {
        printf("Error opening file!");
        return 1;
    }
    while (!feof(original))
    {
        fread(&byte, 1, 1, original);
        fwrite(&byte, 1, 1, backup);
    }
    fclose(original);
    fclose(backup);
    return 0;
}
```

fopen פקודה שיוצרת קובץ בשם כמו הקובץ שהועבר כפרמטר לתוכנית ומוסיפה 2 לסופו(לדוגמא: data.txt2).
wb – פתיחת קובץ בינארי חדש לכתיבה(העותק).

קבצים:

- קיימים 3 קבצי קלט/פלט תקניים הנפתחים ע"י מערכת ההפעלה בתחילת התכנית ומוגדרים להם 3 מצביעים בהתאם:
  1. stdin - מצביע לקובץ הקלט התקני
  2. stdout - מצביע לקובץ הפלט התקני
  3. stderr - מצביע לקובץ השגיאה התקני

פתיחת קובץ

- ראשית מגדירים מצביע לקובץ:

```
FILE* fin; /*input file pointer*/
FILE* fout;/*output file pointer*/
```

- קוראים לפונקציה **fopen**() ומעבירים לה פרמטרים: שם קובץ לפתיחה ומוד הפתיחה

```
fin = fopen("file.dat", "rt");
fout = fopen("file.dat", "wt");
```

- מודי פתיחה בסיסיים
  - המחרוזת "rt" מציינת פתיחה במוד קריאה ובפורמט טקסט, המחרוזת "wt" מציינת פתיחה במוד כתיבה ובפורמט טקסט.
  - ניתן לפתוח במוד לקריאה וכתיבה ("rw").
  - אם לא צויין סוג קובץ - **ברירת המחדל** היא פורמט קובץ טקסט.

קריאה/כתיבה	סוג קובץ
r – read	t - text
w - write	b - binary
a - append	

```
fin = fopen("file.dat", "rwb");
```

- במידה ופעולת הפתיחה/סגירה נכשלת מוחזר NULL ע"י הפונקציה fopen().

פתיחת קובץ סיכום פרמטרים

r – פתיחת קובץ לקריאה (read) בלבד. חובה שיהיה קיים קובץ בשם זה.
w – יצירת קובץ חדש לכתיבה (write). אם כבר קיים קובץ בשם זה – הוא יימחק.
a – פתיחת קובץ להוספה (append). המצביע עומד על סוף הקובץ, ורק שם מותר לכתוב נתונים חדשים. פעולות קריאה אסורות.
r+ - פתיחת קובץ לעדכון (מותרת גם קריאה וגם כתיבה). חובה שהקובץ יהיה כבר קיים.
w+ - יצירת קובץ חדש לכתיבה ולקריאה. אם כבר קיים קובץ בשם זה – הוא יימחק.
a+ - פתיחת קובץ לקריאה ולהוספה. המצביע עומד על סוף הקובץ, ורק שם מותר לכתוב נתונים חדשים. מותר לקרוא נתונים קיימים, אך לא לשנותם.

פרמטר	האם מותר לקרוא?	האם מותר לכתוב?	לאיפה בקובץ מוחזר המצביע?	מה קורה אם כבר קיים קובץ בשם זה?	מה קורה אם לא קיים קובץ בשם זה?
r	כן	לא	תחילת הקובץ	ייפתח	יוחזר NULL
r+	כן	כן	תחילת הקובץ	ייפתח	יוחזר NULL
w	לא	כן	תחילת הקובץ	יימחק	יווצר
w+	כן	כן	תחילת הקובץ	יימחק	יווצר
a	לא	רק בסוף	סוף הקובץ	ייפתח	יווצר
a+	כן	רק בסוף	סוף הקובץ	ייפתח	יווצר

- **בפורמט טקסט** הנתונים נשמרים עפ"י הייצוג המחרוזתי שלהם, כלומר בקוד ה- ASCII של התווים.
- **בפורמט בינרי** לעומת זאת נשמר הנתון בדיוק כפי שהוא מיוצג בזיכרון.
- לדוגמא, המספר ההקסה-דצימלי F123456 יאוחסן בקובץ בפורמט טקסט ע"י 7 תווי ה- ASCII המרכיבים אותו:

70	49	50	51	52	53	54	

  - (כל משבצת מתארת בית בודד בקובץ). 70 הוא קוד ASCII של האות 'F', 49 הוא קוד ASCII של הספרה '1', 50 הוא קוד ה- ASCII של הספרה '2' וכן הלאה.
- בפורמט בינרי המספר F123456 יישמר במערכת כ int - שלם התופס 4 בתים :

56	34	12	0F				

המשך קבצים:

פעולה שמשכפלת את התוכן של קובץ לקובץ אחר.(קובץ טקסט) (לא רלוונטי למבחן)

```
int main()
{
    FILE *fin, *fout;
    int ch;
    fin = fopen("Read.BAT", "rt");
    if (fin == NULL)
    {
        fprintf(stderr, "Cannot open input file.\n");
        return 1;
    }
    fout = fopen("Write.BAT", "wt");
    if(fout == NULL)
    {
        fprintf(stderr, "Cannot open output file.\n");
        return 1;
    }
    while((ch = fgetc(fin))!= EOF)
       fputc(ch, fout);
    fclose(fin);
    fclose(fout);
    return 0;
}
```

fin משמש לפתיחת קובץ טקסט לקריאה

אם פתיחת הקובץ נכשלה, מדפיסים לstderr הודעת כשלון ומסיימים את התכנית

fout משמש לפתיחת קובץ טקסט אחר לכתיבה

קוראים תו תו מהקובץ עד סופו וכותבים תו תו לקובץ פלט

יש לסגור את הקבצים עם סיום העבודה

8

fgets/fputs

- קריאה כתיבה מקובץ טקסט של שורות שלמות ניתן לעשות בקלות על ידי: (לא רלוונטי למבחן)
  - fgets() - פונקציה לקריאת שורת טקסט מהקובץ לתוך מחרוזת.

קובץ טקסט

```
fgets(char *str, int n, FILE *stream)
```

מצביע למחרוזת — מקסימום תווים לקריאה — מצביע לקובץ

  - fputs() - פונקציה לכתיבת מחרוזת לקובץ כשורת טקסט.

```
fputs(char *str, FILE *stream)
```

מצביע למחרוזת — מצביע לקובץ

- שתי הפונקציות מוצהרות בקובץ הספרייה stdio.h ומקבלות כפרמטר מחרוזת

- הקריאה והכתיבה לקובץ טקסט דומה לפעולות המקבילות בקלט/פלט התקני:
  המקבילות לפונקציות printf ו- scanf הן **fprintf** ו- **fscanf**
- הפרמטר הראשון המועבר לfscanf וfprintf הוא המצביע לקובץ.

```
fscanf(fin, "%d %s", &num, str);/*read num and str from file*/
fprintf(fout, "%d %s", num, str);/*write num and str to file*/
```

- הפונקציות fputc ו- fgetc כותבות וקוראות תו בודד אל/מקובץ.

ממשק הפונקציות:

```
int fputc(int c, FILE * fp);
int fgetc(FILE * fp);
```

הפונקציה feof()

- הפונקציה feof מוגדרת ב-stdio.h
- פונקציה זו מקבלת מצביע לקובץ, ומחזירה ערך שונה מאפס כאשר המצביע ביצע פעולת קריאה מעבר לסוף הקובץ, ואת הערך 0 כאשר עוד לא הגענו לסוף קובץ

```
int feof(FILE *stream)
```

- הפונקציה משמשת לבדיקת EOF עבור קבצי טקסט וקבצים בינאריים.

כתיבה וקריאה קבצים בינאריים

- בכתיבה בינרית לקובץ מתבצעת העתקה של תמונת הזיכרון של הנתונים בזיכרון לקובץ.
- קריאה בינרית מבצעת את הכיוון ההפוך.
- קובץ בינארי הוא למעשה אסופה של בתים שקשה לנו לקרוא.
- כתיבה וקריאה בינריים מבוצעות ע"י הפונקציות **fread()** ו- **fwrite()**

```
size_t fread(void* buffer, size_t size, size_t count, FILE* fp);
size_t fwrite(const void* buffer, size_t size, size_t count, FILE* fp);
```

- שתי הפונקציות מקבלות את הפרמטרים:
  - **buffer** - מצביע למאגר הנתונים המיועדים לכתיבה / לקריאה
  - **size** - גודלו של איבר **בודד** במערך בבתים
  - **count** - מספר האיברים : 1 לנתון בודד, למערך - מספר האיברים הנכתבים/נקראים
  - **fp** - מצביע לקובץ
- הפונקציה fwrite מעתיקה מהמקום בזיכרון ש- buffer מצביע עליו size*count בתים לקובץ.
  בדומה, fread מבצעת את הקריאה.
- הערך המוחזר משתי הפונקציות הוא מספר האיברים (count) שנקראו / נכתבו (מטיפוס size_t)

המשך קבצים:

כתיבה לקובץ בינארי - דוגמא

```
FILE *fp;
int  x = 56;
float y = 34.55f;
struct S
{
    int i;
    char str[10];
};
struct S s_arr[3] = {{5,"hello"}, {6,"world"}, {7,"!"}};
/* open file for writing in bin mode */
if( (fp = fopen( "data.out", "wb" )) != NULL )
{
    fwrite(&x, sizeof(int), 1, fp);
    fwrite(&y, sizeof(float), 1, fp);
    fwrite(s_arr, sizeof(struct S), 3, fp);
    fclose(fp);
}
else
{
    puts("could'nt open file!" );
    return;
}
```

פתיחת הקובץ בפורמט כתיבה בינארי 'wb'

כתיבה ע"י fwrite(): בכתיבת משתנה יחיד מציינים את גודלו (sizeof) ו-count =1

סוגרים את הקובץ עם סיום הכתיבה

```
/* open file for reading in bin. mode */
if( (fp = fopen( "data.out", "rb" )) != NULL )
{
    fread(&x, sizeof(int), 1, fp);
    fread(&y, sizeof(float), 1, fp);
    fread(s_arr, sizeof(struct S), 3, fp);
    fclose(fp);
}
else
{
    puts("could'nt open file!" );
    return;
}
printf("x=%d, y=%.2f, s_arr[0].str=%s", x, y, s_arr[0].str);
```

פתיחת הקובץ בפורמט קריאה בינארי 'rb'

קריאה ע"י fread(): בכתיבת משתנה יחיד מציינים את גודלו (sizeof) ו-count =1

סוגרים את הקובץ עם סיום הקריאה

חיפוש בקבצים בינריים – fseek()

- אחת הפעולות השכיחות בטיפול בקבצים בינריים היא חיפוש של רשומה כלשהי.
- הפונקציה **fseek()** מאפשרת להזיז את סמן הקריאה/כתיבה של הקובץ ממקומו הנוכחי למיקום מסויים בקובץ:

```
fseek(<קובץ> ,<היסט> , <נקודת מוצא>)
```

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

1. **fp** - מצביע לקובץ
2. **offset** - מספר הבתים להזזה ביחס ל- origin
3. **origin** - מיקום התחלתי. ערך זה יכול להיות אחד מהשלושה:
   - SEEK_SET - התחלת הקובץ
   - SEEK_CUR - המיקום הנוכחי של סמן הקריאה/כתיבה של הקובץ
   - SEEK_END - סוף הקובץ
- fseek() מחזירה 0 אם הצליחה להזיז את הSEEK_CUR. ערך שונה מאפס אם לא הצליחה

תזוזה בקובץ, פונקציית rewind

- אם נרצה, למשל, להצביע על הרשומה השלישית בקובץ items.bin, אז נכתוב:

```
fptr = fopen("items.bin", "wb");
if (fptr)
        fseek(fptr, 2 * sizeof(item), SEEK_SET);
```

- לאיפה היה מצביע הסמן אם היינו מחליפים את SEEK_SET ב-SEEK_CUR?
- אם נרצה להזיז את המצביע לסוף הקובץ אז נכתוב:

```
fseek(fptr, 0, SEEK_END);
```

- ב-stdio.h מוגדרת פונקציה בשם rewind, המקבלת מצביע לקובץ, ומעבירה אותו אל ראש הקובץ. לדוגמא: שתי ההוראות הבאות שקולות לגמרי -

```
rewind(fptr);
fseek(fptr, 0, SEEK_SET);
```

המשך קבצים בינאריים:

fseek() דוגמה

- לדוגמא, נתון קובץ בינרי של רשומות מטיפוס Student הכולל מספר מזהה, שם וממוצע ציונים:

```
typedef struct
{
    int ID;
    String name;
    double average;
} Student;
```

- בקובץ מערך רשומות סטודנטים ממויינות עפ"י מספר הזהות:

הקובץ כולל 10 רשומות. סמן הקריאה/כתיבה של הקובץ נמצא לאחר הרשומה החמישית ולפני הרשומה השישית.

- ניתן להזיז את הסמן לרשומה השלישית ע"י

```
fseek(file, 2 * sizeof(Student), SEEK_SET);
```

- הסבר: הפונקציה fseek מזיזה את סמן הקריאה/כתיבה ביחס לראשית הקובץ (SEEK_SET) למיקום 2*sizeof(Student) שהוא מקום התחלת הרשומה השלישית.

---

fseek() המשך דוגמה

- ניתן לקרוא את הרשומה מהקובץ ע"י **fread** ולהדפיס את ערכיה.
- לדוגמא, הפונקציה הבאה מזיזה את הסמן לרשומה התשיעית (אינדקס 8), קוראת ומדפיסה את ערכי הרשומה:

```
void f(FILE* file)
{
    Student s;
    fopen("file.dat", "rb");
    fseek(file, 8 * sizeof(Student), SEEK_SET);
    fread(&s, sizeof(Student), 1, file);/* read record */
    printf("\nRecord %d name is %s, avg=%f\n", s.ID, s.name, s.average);
}
```

```
Record 38 name is Yfat, avg=8.500000
```

הפונקציה ftell()

- מחזירה את מיקום סמן הקובץ בבתים (bytes) ביחס לתחילת הקובץ (SEEK_SET)

```
long int ftell(FILE* stream)
```

- משמשת לחישוב גודל קובץ:

```
/* calc file size */
fseek(file, 0, SEEK_END);  /* move to end of file */
file_size = ftell(file);
```

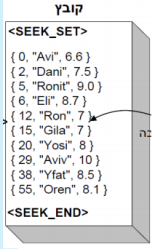
דוגמא - bin search לפי ID בקובץ בינארי

- הפונקציה הבאה מבצעת חיפוש בינרי של רשומה בקובץ. הפונקציה מקבלת כפרמטרים מצביע לקובץ ומספר מזהה לחיפוש:

```
void bin_search(FILE *file,  int ID)
{
    . . .
    /* do binary search */
    min = 0;
    max = file_size/sizeof(Student) - 1;
    for(found = FALSE; min <= max; )
    {
        avg = (max + min)/2;
        fseek(file, avg*sizeof(Student), SEEK_SET); /* move to the record avg */
        fread(&s, sizeof(Student), 1, file); /* read record */
        printf("%d ", s.ID);
        if(s.ID==ID)
        {
            printf("\nRecord %d found: name is %s",s.ID, s.name);
            found = TRUE;
            break;
        }
        else
            if(ID < s.ID) max = avg - 1;
            else min = avg + 1;
    }
```

בלולאה מבצעים את החיפוש הבינרי: בכל חזרה, מחשבים את החציון, avg ומשווים את ID שמחפשים עם זה של הרשומה שבחציון. בהתאם לכך ממשיכים לחפש בחציון העליון, או התחתון:

בכל חזרה מזיזים את min או max לכיוון החציון (עפ"י תוצאת ההשוואה) עד למציאת הרשומה, או עד ש- min ו- max בעלי ערך זהה, מה שמציין שהרשומה שמחפשים לא קיימת בקובץ.



קריאה לחיפוש בינארי(bin_search)

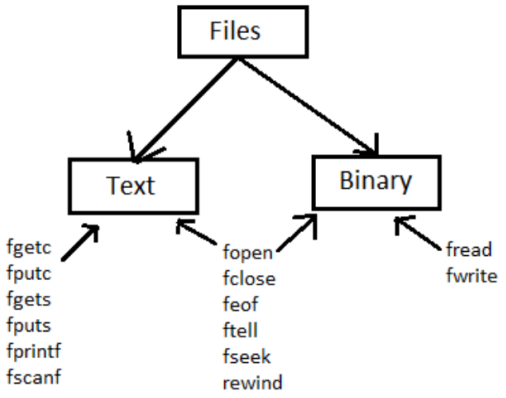
```
/* open for reading */
if( (file = fopen( file_name, "rb" )) != NULL )
{
    /* search for some IDs */
    bin_search(file, 2);
    bin_search(file, 0);
    bin_search(file, 55);
    bin_search(file, 103);
    fclose( file );
}
else
    printf( "Problem opening the file %s\n", file_name );
```

```
Searching for record 2 avg = 12 avg = 2
Record 2 found: name is Dani
Searching for record 0 avg = 12 avg = 2 avg = 0
Record 0 found: name is Avi
Searching for record 55 avg = 12 avg = 29 avg = 38 avg = 55
Record 55 found: name is Oren
Searching for record 103 avg = 12 avg = 29 avg = 38 avg = 55
Record 103 not found!
C:\Users\Administrator\source\repos\files\Debug\files.exe (process 20016) exited with code 0.
```

המשך קבצים בינאריים:

סיכום – פונקציות גישה לקבצי טקסט וקבצים בינאריים

- הפונקציות לפתיחת וסגירת קובץ הן **fopen** ו- **fclose** בהתאמה.
- הפונקציות לקריאה וכתיבה בינריים הן **fread** ו- **fwrite**.
- הפונקציות לכתיבה וקריאה במוד טקסט הן פונקציות מקבילות לפונקציות הקלט/פלט התקניות בתוספת האות f בתחילתן: **fscanf**, **fprintf**, **fgetc** וכו'.
- הפונקציה **fseek** משמשת להזזת סמן הקריאה/כתיבה של הקובץ למיקום מסויים.
- הפונקציה **ftell** מחזירה את מרחק הסמן מתחילת הקובץ בבתים.
- **rewind** מחזירה את הסמן לתחילת הקובץ ו**feof** מחזירה TRUE אם הגענו לסוף הקובץ



פעולות על סיביות(Bitwise):

- ישנן מספר פעולות על סיביות:
  - פעולת AND המסומנת על-ידי האופרטור &.
  - פעולת OR המסומנת על-ידי האופרטור |.
  - פעולת NOT המסומנת על-ידי האופרטור ~.
  - פעולת XOR המסומנת על-ידי האופרטור ^ .
  - פעולת הזזה שמאלה המסומנת על-ידי האופרטור << .
  - פעולת הזזה ימינה המסומנת על-ידי האופרטור >> .
- שלושה מהאופרטורים הפועלים על סיביות הם אופרטורים בינאריים (המצפים לקבל שני אופרנדים),
- רק האופרטור NOT הוא אופרטור אונארי (מקבל אופרנד אחד).

AND:

```
int a = 72 & 184;
printf(" % d", a);
```

```
01001000
10111000   &
00001000
```

הפלט יהיה 8.

0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

המסקנה היא שניתן להשתמש בפעולה & על מנת למסך (to mask) סיבית מסוימת, ולבודד אותה משאר הייצוג של המספר.

:OR

```
01001000
|
10111000
11111000
```

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

הפלט יהיה 248.

```
int a = 72 | 184;
printf(" % d", a);
```

המסקנה היא שניתן להשתמש בפעולה | על מנת להדליק סיביות מבוקשות, מבלי לשנות סיביות אחרות.

דוגמא של שניהם :

מה יהיה הפלט של קטע הקוד הבא:

```
int num = 11;
int a = num | num, b = num & num;
printf(" % d % x % o\n", a, a, a);
printf(" % d % x % o\n", b, b, b);
```

```
11 b 13
11 b 13
```

```
0 -1
```

הסבר:

גם a וגם b הם 11.

d% זה המספר בבסיס 10(דצימלי)

x% זה המספר בבסיס 16(הקסהדצימלי)

o% זה המספר בבסיס 8(אוקטבי)

המשך פעולות על סיביות(Bitwise):

:NOT

```
~01001000
 10110111
```

~0 = 1
~1 = 0

הפלט יהיה -73.

```
short int a = 72;
printf(" %hd", ~a);
printf(" %hu", ~a);
```

ומה יהיה הפלט אם נרשום %hu במקום %hd?

הפלט יהיה 65,463 (65,463 + |-73| = 65,536=2^{16})

hd – signed short int

hu – unsigned short int

המס ה | על מנת להדליק סיביות מבוקשות, מבלי לשנות סיביות אחרות.

- אם a ו-b הן סיביות, אז מתקיימים גם חוקי דה-מורגן:
  - ~(a & b) == ~a | ~b
  - ~(a | b) == ~a & ~b

עוד דוגמא :

```
char a = 13;
printf(" % d % d", a & ~a, a | ~a);
```

הפלט :

```
0 -1
```

XOR: שמפעילים על סיבית מסוימת ^ עם 0 היא לא משתנה, וכאשר מפעילים על סיבית מסוימת ^ עם 1 היא מתהפכת

```
0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0
```

דוגמא נוספת:

```
01001000
        ^
10111000
11110000
```

הפלט יהיה 240.

```
unsigned short int a = 72 ^ 184;
printf(" %hu", a);
```

```
short int a = 72;
short int b = a ^ -1;
printf(" %hd %hu", b, b);
```

```
-73 65463
```

דוגמא נוספת:

מה יהיה הפלט של קטע הקוד הבא:

```
int num = 11;
int a = num ^ num, b = num ^ ~num;
printf(" %d %d", a, b);
```

הפלט:

```
0 -1
```

הפעולה swap בעזרת שימוש בXOR:

```
void swap(int* a, int* b)
{
    *a ^= *b;
    *b ^= *a;
    *a ^= *b;
}
```

הפלט: (2^2)*7 = 28

הזזה

- בשפת C קיימים אופרטורים המאפשרים להזיז את סיביות הביטוי ימינה ושמאלה.
- האופרטור >> מבצע הזזה סיביות שמאלה, מימין נכנסים אפסים shl
- האופרטור << מבצע הזזה סיביות ימינה. משמאל נכנסים אפסים. shr
- שני האופרטורים מקבלים שני אופרנדים:
  - האופרנד השמאלי זהו הביטוי שאת סיביותיו יש להזיז.
  - האופרנד הימני זהו מס' ההזזות שצריך לבצע.
- מה יהיה הפלט של קטע הקוד הבא?

```
printf(" %d", 7 << 2);
```

הפלט: 7*(2^2) = 28

- מה מבצע קטע הקוד הבא?

```
int a, count = 0;
scanf(" % d", &a);
while (a != 0)
{
    count += a & 1;
    a = a >> 1;/* a >>= 1/* :אפשר היה לרשום בקיצור
}
printf(" % d", count);
```

קטע הקוד מחזיר לי את מספר הסיביות במספר שהתקבל.

המשך פעולות על סיביות(Bitwise):

הזזה:

- מה לדעתכם יהיה הפלט של קטע התכנית הבא?

```
printf("\n %x", -1 << 4);
```

```
fffffff0
```

- ניתן להשתמש באופרטורים של הזזה על מנת לממש ביעילות כפל בחזקות של שתיים או חלוקה בחזקות של שתיים:
  - a = a << k יכפיל את a פי 2^k.
  - a = a >> k יחלק את a פי 2^k.

כל 4 הזזות זה בית

$$a \ll n = a \times 2^n$$

$$a \gg n = \left\lfloor \frac{a}{2^n} \right\rfloor$$

מצביעים לפונקציה:

```
int main()
{
    int (*array[])(int, int) = {subtract, multiply, divide};
    int product = (*array[1])(3,15);

    printf("product: %d\n", product);
```

void יכול להיות) פלט פונקציה

הפרמטרים שהפונקציה מקבלת

הפעולות עצמן

מערך מצביעים לפונקציות השונות

```
product: 45 = 3*15
Program ended with exit code: 0
```

:memory.h

memcpy

```
void * memcpy ( void * destination, const void * source, size_t num );
```

הפונקציה מעתיקה num בתים מsource לdestination.

```
const char src[50] = "bye!bye";
char dest[50];
strcpy(dest,"Heloooo!!");
printf("Before memcpy dest = %s\n", dest);
memcpy(dest, src, 5);
printf("After memcpy dest = %s\n", dest);
```

```
Before memcpy dest = Heloooo!!
After memcpy dest = bye!boo!!
```

memcmp

```
int memcmp ( const void * ptr1, const void * ptr2, size_t num );
```

הפונקציה משווה את הnum בתים הראשונים בptr1 לnum בתים הראשונים בptr2.(אם הם זהים מחזיר 0)

if Return value < 0 then it indicates str1 is less than str2.

if Return value > 0 then it indicates str2 is less than str1.

if Return value = 0 then it indicates str1 is equal to str2.

```
memcpy(str1, "abcdef", 6);
memcpy(str2, "ABCDEF", 6);

ret = memcmp(str1, str2, 5);
```

memchr

```
const void * memchr ( const void * ptr, int value, size_t num );
```

הפונקציה בודקת האם value נמצא בnum בתים הראשונים בptr. במידה ונמצא מופע אז מוחזר המצביע לתחילת המופע, במידה ולא מוחזר מצביע NULL.

```
char str[] = "ABCDEFG";
char ch = 'D';
char *ps = memchr(str, ch, strlen(str));
printf("character %c found: %s\n", ch, ps);
```

```
character D found: DEFG
```

memset

```
void * memset ( void * ptr, int value, size_t num );
```

הפונקציה ממלא את הnum בתים הראשונים בptr בערך value ומחזירה מצביע על תחילת ptr.

```
char str[50];
strcpy(str,"helloWorld");
puts(str);
memset(str,'$',7);
puts(str);
```

```
helloWorld
$$$$$$$rld
```

memmove

```
void * memmove ( void * destination, const void * source, size_t num );
```

הפונקציה מעבירה את הnum בתים הראשונים מsource לdestination.

יותר בטוח להשתמש בmemmove כאשר עושים קופי בoverlapping (כאשר מכניסים ערך של בית על בית מסוים כך שזה אותו אחד, למשל: Start אם נעתיק את החלק art להתחלה אז אותו הa מועתק). הmemmove יותר בטוח משום שהוא משתמש בbuffer ומעביר אליו את המידע ורק אז מעביר אותו לdestination לעומת הmemcpy שישר מעתיק לdestination (יעשוי אותה פעולה לרוב אך תלוי בcompiler).

לפניכם הפונקציה הרקורסיבית ChangeDigits . הפונקציה מקבלת מספר שלם חיובי n , שמספר ספרותיו הוא זוגי, ומחזירה מספר חדש שבו הוחלפו מיקומי הספרות במספר n , בין הספרות שבמקומות הזוגיים לאלו שבמקומות האי–זוגיים (לדוגמה, הפונקציה הופכת את המספר 346679 למספר 436697) .

בקוד הפעולה חסרים **שני** ביטויים, המסומנים במספרים בין סוגריים עגולים. רשום בדף התשובות את מספרי הביטויים החסרים (1)–(2) בלבד

```
int ChangeDigits(int num)
{
    if (num < 100)
        return (num%10)*10 + num/10;
    return (num % 10) * 10 + (num / 10)%10 + ChangeDigits(num/100) *100 ;
}
```

File handling functions	Description
fopen ()	fopen () function creates a new file or opens an existing file.
fclose ()	fclose () function closes an opened file.
getw ()	getw () function reads an integer from file.
putw ()	putw () functions writes an integer to file.
fgetc ()	fgetc () function reads a character from file.
fputc ()	fputc () functions write a character to file.
gets ()	gets () function reads line from keyboard.
puts ()	puts () function writes line to o/p screen.
fgets ()	fgets () function reads string from a file, one line at a time.
fputs ()	fputs () function writes string to a file.
feof ()	feof () function finds end of file.
fgetchar ()	fgetchar () function reads a character from keyboard.
fprintf ()	fprintf () function writes formatted data to a file.
fscanf ()	fscanf () function reads formatted data from a file.
fputchar ()	fputchar () function writes a character onto the output screen from keyboard input.
fseek ()	fseek () function moves file pointer position to given location.
SEEK_SET	SEEK_SET moves file pointer position to the beginning of the file.
SEEK_CUR	SEEK_CUR moves file pointer position to given location.
SEEK_END	SEEK_END moves file pointer position to the end of file.
ftell ()	ftell () function gives current position of file pointer.
rewind ()	rewind () function moves file pointer position to the beginning of the file.

getc ()	getc () function reads character from file.
getch ()	getch () function reads character from keyboard.
getche ()	getche () function reads character from keyboard and echoes to o/p screen.
getchar ()	getchar () function reads character from keyboard.
putc ()	putc () function writes a character to file.
putchar ()	putchar () function writes a character to screen.
printf ()	printf () function writes formatted data to screen.
sprinf ()	sprinf () function writes formatted output to string.
scanf ()	scanf () function reads formatted data from keyboard.
sscanf ()	sscanf () function Reads formatted input from a string.
remove ()	remove () function deletes a file.
fflush ()	fflush () function flushes a file.

File operation	Declaration & Description
fseek() עובר עם הסמן לאוביקט הבא	Declaration: int fseek(FILE *fp, long int offset, int whence) fseek() function is used to move file pointer position to the given location. where, fp – file pointer offset – Number of bytes/characters to be offset/moved from whence/the current file pointer position whence – This is the current file pointer position from where offset is added. Below 3 constants are used to specify this.
SEEK_SET	SEEK_SET – It moves file pointer position to the beginning of the file.
SEEK_CUR	SEEK_CUR – It moves file pointer position to given location.
SEEK_END	SEEK_END – It moves file pointer position to the end of file.

לדוגמא מעבר לאוביקט השלישי
עושים fseek
לשני
0
1
2
3

הפונקציה qsort()

```
int cmpfunc (const void * a, const void * b) {
   return ( *(int*)a - *(int*)b );
}
int values[] = { 88, 56, 100, 2, 25 };
qsort(values, 5, sizeof(int), cmpfunc);
```

```
Before sorting the list is:
88 56 100 2 25
After sorting the list is:
2 25 56 88 100
```