האוניברסיטה הפתוחה

מעבדה בתכנות מערכות

מדריך למידה

מיכל אבימור

מדריך למידה לספר

The C Programming Language | Second Edition

Brian. W. Kernighan, Dennis. M. Ritchie

תנאי שימוש בקובץ הדיגיטלי:

1. הקובץ הוא לשימושך **האישי** בלבד. פרטים מזהים שלך מוטבעים בקובץ בצורה גלויה ובצורה סמויה.

2. השימוש בקובץ הוא אך ורק למטרות לימוד, עיון ומחקר אישי.

3. העתקה או שימוש בתכנים נבחרים מותרת בהיקף העומד בכללי השימוש ההוגן, המפורטים בסעיף 19 לחוק זכות יוצרים 2007. במקרה של שימוש כאמור חלה חובה לציין את מקור הפרסום.

4. הנך רשאי/ת להדפיס דפים מחומר הלימוד לצורכי לימוד, מחקר ועיון אישיים. אין להפיץ או למכור תדפיסים כלשהם מתוך חומר הלימוד.

האוניברסיטה הפתוחה

מעבדה בתכנות מערכות

מדריך למידה

מיכל אבימור

מדריך למידה לספר

The C Programming Language

Second Edition

Brian.W. Kernighan

Dennis. M. Ritchie

20465
מהדורה פנימית
לא למכירה ולא להפצה
מק"ט 20465-5054

צוות הקורס

מיכל אבימור – כותבת

פרופ׳ ראובן אביב – אחראי אקדמי

ד״ר יצחק הרץ

חוה ניומן – עורכת

הדפסה מתוקנת – ספטמבר 2020

בית ההוצאה לאור של האוניברסיטה הפתוחה, הקריה ע״ש דורותי דה רוטשילד, דרך האוניברסיטה 1, ת״ד 808, רעננה 4353701.
The Open University of Israel, The Dorothy de Rothschild Campus, 1 University Road, P.O.Box 808, Raanana 4353701.
Printed in Israel.

תוכן העניינים

הקדמה

הקורס "מעבדה בתכנות מערכות" מבוסס על תכנות בשפת C. מטרתנו היא לאפשר, לאחר סיום המעבדה, לתכנת ברמה מעמיקה בשפת C, על פי עקרונות בסיסיים של הנדסת תוכנה. שליטה בשפה תבטיח לכם את האפשרות לתרגל בנושאים אחרים, כמו אלו העוסקים במבני נתונים או באלגוריתמים, אשר בהם יש חשיבות גדולה ליישום.
תפקידו של מדריך למידה זה הוא לסייע לכם בלימוד שפת C. ספר הלימוד של השפה הוא:

The C Programming Language
Second Edition
Brian.W. Kernighan
Dennis.M. Ritchie

מדריך זה מכיל הסבר נוסף לגבי החומר שבספר הקורס, הנחיות לגבי סדר הלימוד, דוגמאות המדגישות תכונות אופייניות של השפה, תזכורות מקורסים קודמים וכן פתרונות לתרגילים המופיעים בספר. פרקי המדריך ילוו למעשה את הקריאה בספר הלימוד ולכן שמות הפרקים במדריך זהים לשמות הפרקים בספר הלימוד. גם הסעיפים במדריך ומספורם חופפים לאלה שבספר. אנו ממליצים לכם לקרוא את הסעיפים שבמדריך במקביל לאלה שבספר הלימוד, אלא אם כן הסעיף שבמדריך עומד בפני עצמו ואינו מקביל לסעיף כלשהו מתוך ספר הלימוד.

התרגול מהווה חלק בלתי נפרד מעצם הלימוד של כל שפת תכנות, לכן אנו ממליצים לכם להריץ את התכניות המופיעות בספר, ולפתור בעצמכם את התרגילים שבמדריך ואת המטלות. פתרון התרגילים מקנה כלים להתמודדות עם פרויקט התכנות הנלווה לקורס.

שפת C היא שפה פופולרית מאוד והשימוש בה רב. למרות היותה שפה ותיקה, היא משמשת עד היום הודות ליעילות ולשליטה המרבית המתאפשרת למתכנת באמצעותה. בשפה זו ניתן לקרוא ולכתוב במערכות רבות: במערכות מדעיות מתקדמות, במערכות הפעלה מודרניות הכתובות בשפת C, וכמובן שגם בשפת C++.

המעבר לשפת C++, שהיא שפה מונחית עצמים, יכול להיעשות בקלות רבה יותר לאחר רכישת הידע בשפת C.

שפת C קשורה קשר הדוק למערכת ההפעלה UNIX. מערכת הפעלה זו נכתבה ב-C. כמו כן, יש לתכניות שירות מסוימות של UNIX פקודות מיוחדות המיועדות לתמוך במיוחד ב-C, כגון indent, ctags.

ה-Reference-Manual שימש במהדורה הראשונה של הספר כהגדרה למבנה השפה (מבחינת תחביר וסמנטיקה). בשנת 1983 כינס מכון התקנים האמריקני – ANSI – ועדה שמטרתה הייתה כתיבת הגדרה מחודשת ל-C, כך שהשפה תהיה "נקייה" במידת האפשר מתלות במכונה שעבורה נכתב המהדר (machine independent), שכן ההגדרה הראשונה לא הייתה נקייה מתלות כזאת. הדבר נועד להבטיח שתוכנה הנכתבת במחשב מסוים תהיה ניידת (portable). כלומר, ניתן יהיה להריצה במחשב אחר, בעל ארכיטקטורה אחרת, בלא שיהיה כל צורך לשנותה.

הספר מתייחס להגדרה אשר יצאה מתחת ידה של ועדה זו כ- The ANSI Standard for C. גרסה זו כבר אושרה רשמית, אך לא כל מהדר של C תומך בה. לכן, כאשר משתמשים במהדר, יש לוודא תחילה כי הוא תומך בהגדרה התקנית.

סביבת העבודה שתשמש אותנו לתכנות בקורס זה מתוארת בנפרד. אולם מובן שמדריך זה יכול לשמש לכל סביבת תכנות, בהתאם לעקרונות הכתיבה של קוד נייד (portability), שאליהם נתייחס גם בהמשך.

פרק 1. A Tutorial Introduction

פרק זה מציג סקירה כללית של השפה, וככזה הוא עמוס בפרטים. כל הנושאים נדונים שוב גם בפרקים הבאים. לכן, בשלב זה אין צורך לזכור את כל הפרטים, אך חשוב להבין כבר עתה כיצד כותבים תכנית כללית ב-C וכיצד מבצעים קלט/פלט פשוט.

יש לשים לב: בפרק זה מוסבר רק כיצד לכתוב תכנית בשפה. כדי לדעת כיצד לערוך אותה במעבד, כיצד להעביר הידור וליצור קובץ הרצה, יש לקרוא את יחידת הלימוד "מדריך לסביבת UNIX".

1.1 התחלה – Getting Started

בקטע זה מוצגת הפונקציה printf, מתוך "הספרייה הסטנדרטית". "הספרייה הסטנדרטית" אינה חלק משפת C עצמה. זהו אוסף של קבצים המכילים הגדרות ופונקציות, לשימוש פעולות שכיחות, כגון קלט/פלט. תקן ANSI מגדיר במדויק את פונקציות הספרייה הנ"ל, כך שניתן יהיה להשתמש בפונקציות אלו בכל מערכת תקנית המכילה את שפת C. מכאן נובע שכל מערכת תקנית יכולה לממש את פונקציות הספרייה הסטנדרטית בדרך שונה.
ככלל, שירותי קלט ופלט, כגון הדפסה, טיפול במחרוזות תווים, ניהול זיכרון, אינם חלק משפת C, אלא נכללים בספרייה הסטנדרטית.

כל פונקציות הספרייה הסטנדרטיות מתוארות בנספח B של הספר. תיאור של פונקציה כולל את הפרמטרים שהפונקציה מקבלת, את הערך שאותו היא מחזירה ואת פעולתה.

דוגמה

נראה כאן מימוש אפשרי לפונקציית הספרייה strlen, אשר מתוארת בנספח B בעמוד 250. הפונקציה מקבלת מצביע אל מחרוזת תווית, ומחזירה את אורכה. על פי דרישות התקן, אב הטיפוס לפונקציה זו צריך להפיע בקובץ <string.h>. (בשלב זה אין צורך להעמיק בדוגמה זו. הדוגמה תובן טוב יותר, לאחר קריאת פרק 4, העוסק בפונקציות).

```
int strlen(char *s)
{
   int len=0;
   while (*s++)   // while the string did not come to its end
      len++;
   return(len);
}
```

1.2 משתנים וביטויים אריתמטיים
Variables and Arithmetic Expressions

את התחומים (ranges) של טיפוסי הנתונים ניתן למצוא בקובץ limits.h.

יש לשים לב לכך שתו יחיד מציין את אופרטור החילוק (/), הן של שלמים והן של מספרים ממשיים. ההחלטה איזה סוג של חילוק יבוצע תלויה בהקשר (context) שבו הוא מופיע: אם לפחות אחד האופרנדים הינו ממשי, יבוצע חילוק בין מספרים ממשיים. אחרת יבוצע חילוק בין שלמים. הכללים המדויקים יפורטו בפרק 2.
לדוגמה:

```
int x = 3, y = 2;
```

במקרה של החילוק x / y יבוצע חילוק בין שלמים. תוצאת החילוק תהיה 1.

אולם במקרה הבא תוצאת החילוק x / y תהיה 1.5:

```
float x = 3.0, y = 2.0;
```

גם במקרה הבא, תהיה תוצאת החילוק 1.5:

```
int x = 3;
float y = 2.0;
```

בתרגילים הבאים נראה כיצד מושפעת תכנית בשפת C על ידי שינוי סוג המשתנה מ-float ל-int:

תרגיל 1

יש להריץ את התכנית השנייה בסעיף זה כפי שהיא (התכנית בעמוד 12 בספר).

תרגיל 2

יש להריץ את התכנית שוב, אולם הפעם יש להגדיר את המשתנה celsius כ-int (יש לשנות גם את פורמט ההדפסה שלו ב-printf להדפסה של שלם: %d). על התכנית להיראות כך:

```
#include <stdio.h>
main( )
{
    float fahr;
    int celsius; /* type change of celsius */
    int lower, upper, step;
    lower = 0;
    upper = 300;
    step = 20;
    fahr = lower;
    while (fahr <= upper)
    {
        celsius = (5.0 / 9.0) * (fahr - 32.0);
/* Note that in the above line the lhs (Left Hand Side) of the assignment is
of type int and the rhs (Right Hand Side) is of type float */
        printf ("%3.0f %6f\n", fahr, celsius);
        /* change of format */
        fahr = fahr + step;
    }
}
```

הערה: התכנית כוללת הצבה מיידית של קבועים, כפי שמופיע גם בספר. בהמשך הקורס נראה כיצד להימנע מכך, ולאפשר לתכנית להיות קריאה יותר ונוחה לשינויים עתידיים.

נשים לב למשפט ההשמה של המשתנה celsius בתוך לולאת ה-while שבדוגמה למעלה: אגפו הימני הוא מטיפוס float (מספר ממשי), בעוד שאגפו השמאלי הוא integer. המשמעות המתקבלת היא של פונקציית קטיעה (trunc). ההפיכה מטיפוס float לטיפוס int נעשתה בצורה סתומה (לא מפורשת). ב-C אין צורך בפונקציה מיוחדת על מנת לעבור מטיפוס float לטיפוס int. עצם ההשמה קבעה את שינוי הטיפוס. נושא זה יורחב בפרק 2.

תרגיל 3

יש לשנות בתכנית המקורית את פורמט ההדפסה של המשתנה fahr להדפסה של מספר שלם (ללא שינוי בטיפוס של fahr), ולהריץ אותה. על התכנית להיראות כך:

```
#include <stdio.h>
main( )
{
    float fahr, celsius;
    int lower, upper, step;
    lower = 0;
    upper = 300;
    step = 20;
    fahr = lower;
    while (fahr <= upper)
    {
        celsius = (5.0 / 9.0) * (fahr - 32.0);
        printf ("%d %6.1f\n", fahr, celsius);
        /* change of format */
        fahr = fahr + step;
    }
}
```

לאחר הריצה נראה כי השינוי משפיע גם על ההדפסה של המשתנה celsius, למרות שבפורמט ההדפסה שלו לא נעשה כל שינוי. המסקנה היא שהמהדר **לא** בודק התאמה בין טיפוס המשתנה לבין תיאור הדפסתו בפקודת printf. המהדר גם לא מוודא שמספר הערכים המודפסים במשפט printf זהה למספר המשתנים המתואר במחרוזת הפורמט.

1.3 The For Statment

תרגיל 4

יש לבצע את תרגיל 5-1 בספר:

יש לשנות את תכנית המרת הטמפרטורות, כך שתדפיס את הטבלה בסדר הפוך, כלומר מ-300 מעלות עד ל-0.

פתרון

```
#include <stdio.h>
/* print Fahrenheit-Celsius table in reverse order */
/* from 300 down to 0 with decremental step of 20 */
main ( )
{
    int fahr;
    for (fahr = 300; fahr >= 0; fahr = fahr – 20)
        printf("%3d %6.1f\n", fahr, (5.0 / 9.0) * (fahr – 32));
}
```

1.4 Symbolic Constants – קבועים

באמצעות הוראת define# ניתן להגדיר קבועים עבור התכנית.

התו '#' של הוראת define חייב להופיע בעמודה הראשונה של התכנית. שגיאה נפוצה היא להוסיף ';' בסוף משפט ה-define#, דבר הגורם לשגיאות תחביר בעת הידור

התכנית. כשנלמד (למשל בפרק 4) על ה-C pre-processor (שהוא השלב הראשון, לפני שלב ההידור), נבין מדוע יש בכך משום שגיאה וכיצד אפשר לאתר אותה.

מקובל לכתוב את הקבועים הסימבוליים באותיות גדולות, כדי להבדילם ממשתנים רגילים. יש לשים לב ששפת C רגישה לגודל האותיות (Case Sensitive), כלומר השפה מבדילה בין אותיות קטנות לגדולות. כך למשל המשתנה sum שונה מהמשתנה SUM, או Sum.

1.5 Character Input and Output

פונקציות הקלט והפלט של הספרייה אחראיות לטיפול בנתונים, הממומשים כרצף תווים (למעשה, כאוסף של מספרים בינאריים). כך, למשל, יחולקו הנתונים לשורות, כל שורה תסתיים בתו מיוחד, לסימון סוף/תחילת שורה חדשה. המשתמש בספרייה אינו אחראי ואינו צריך להכיר את תצוגת התווים מחוץ לתכנית. מערכות הפעלה שונות יכולות להציג שורה חדשה על ידי רצף שונה של תווים: שורה חדשה יכולה להיות מיוצגת על ידי CR (ASCII 13) ואחר כך LF (ASCII 10) או על ידי סדרת תווים אחרים, או תו אחד בלבד (תרגיל מתקדם: בדיקה כיצד מיוצג הדבר במערכת שלנו). פונקציות הספרייה מבצעות תרגום של סדרת תווים זו לתו '\n' בקריאה, ותרגום של '\n' לסדרה זו בכתיבה. התרגום הוא שקוף עבורנו, אולם יש להיות מודעים לקיומו.

נזכיר כי תיאור של כל פונקציות הספרייה הסטנדרטיות מופיע בנספח B בספר.

1.5.1 File Copying

הפונקציה getchar יכולה לקרוא את התו הבא מלוח המקשים, בעוד putchar יכולה להציג את התו על המסך.
נסביר מדוע טיפוס המשתנה c, שבדוגמה בסעיף זה, הוכרז כ-int ולא כ-char:
אילו היה c מוכרז כ-char, אזי גודל הזיכרון שהיה מוקצה עבורו בסביבת ה-UNIX שלנו היה בן בית אחד (8 סיביות). לעומת זאת, ערכו של הקבוע EOF הוא 1– (נדגיש כי ערך זה הוא 1–, ברוב המקרים. ייתכן שבמכונה אחרת ייקבע ל-EOF ערך אחר. את ערכו של הקבוע EOF ניתן לראות בקובץ stdio.h). אם המעבד מייצג מספרים בשיטת

המשלים ל-2, מיוצג המספר 1– על-ידי 0xFF (0x היא הקידומת ב-C לכתיבת מספר הקסדצימלי). קידוד זה מייצג גם את התו שערכו 255 (לא בכל מכונה חייבת להיות הרחבה של תווי ה-ASCII לקוד מעל 127). אם מופיע בקובץ תו שערך ה-ASCII שלו הוא 255, אזי תוצאת התנאי במשפט ה-while תהיה true, ולכן במקרה זה נצא מהלולאה לפני שהגענו לסוף הקובץ. אולם, כאשר c הוא מטיפוס int, EOF מיוצג (בשיטת המשלים ל-2) על ידי 0xFFFF, ואילו 255 מיוצג על ידי 0x00FF, וההשוואה ביניהם תיתן false.

תזכורת

שיטת המשלים ל-1 (one's complement)

כדי לייצג מספר בינארי שלילי, מוצאים תחילה את הייצוג הבינארי של הערך המוחלט, ואז משלימים כל סיבית במספר ל-1. כל סיבית 0 הופכת ל-1, וכל סיבית 1 הופכת ל-0. כלומר לפי שיטה זו, הסיבית השמאלית של מספר חיובי תהיה 0, והסיבית השמאלית של מספר שלילי תהיה 1.

דוגמה

הייצוג של "6-" בחמש סיביות הוא 11001, על פי השלבים:

א. ייצוג הערך המוחלט "6": 00110

ב. הפיכת הסיביות: 11001

שיטת המשלים ל-2 (two's complement)

שני השלבים הראשונים בשיטה זו זהים לשיטת המשלים ל-1, לאחר מכן מוסיפים 1 למספר שקיבלנו. כך שמבצעים מעין השלמה של הסיבית הימנית ביותר ל-2. גם בשיטה זו הסיבית השמאלית מייצגת את סימן המספר.

דוגמה

הייצוג של "6-" בחמש סיביות הוא 11010, על פי השלבים:

א. ייצוג הערך המוחלט "6": 00110

ב. הפיכת הסיביות: 11001

ג. הוספת 1: 11010

תרגיל 5

א. יש לבצע את תרגיל 6-1 שבספר:

לוודא כי הביטוי getchar() != EOF הוא 0 או 1.

פתרון

הדפסת הערך הנ"ל:

```
#include <stdio.h>
main ( )
{
    int c;
    while (c = (getchar ( ) != EOF))
        printf ("%d\n", c);
/* As long as getchar didn't return EOF c should get value of 1 */
    printf ("at EOF c is ");
    printf ("%d\n", c);
/* the above two lines can be assembled together by writing printf ("at EOF
c is %d\n", c; */
}
```

ב. יש לבצע את תרגיל 1-7 שבספר: יש לכתוב תכנית המדפיסה את ערך EOF.

פתרון

```
#include <stdio.h>
main ( )
{
    printf ("The value of  EOF is %d\n", EOF);
}
```

Character Counting 1.5.2

יש לשים לב שהקשת <enter> בעת הרצת התכניות גורמת להגדלת nc באחד.
כמו כן שימו לב שהתנאי להפסקת לולאת ה-for בתכנית השנייה אינו תלוי בערך משתנה הלולאה (nc). תיאור מלא של לולאת for יובא בפרק 3.

Line Counting 1.5.3

תרגיל 6

א. יש לבצע את תרגיל 8-1 שבספר:

לכתוב תכנית לספירת רווחים, טאבים ושורות חדשות.

פתרון

```
#include <stdio.h>
main ( )
{
    int blank_cnt;                  /* counter for blanks */
    int tab_cnt;                    /* counter for tabs */
    int newline_cnt;                /* counter for newlines */
    int read_char;      /* The char value we get from the input */
    blank_cnt = 0;
    tab_cnt = 0;
    newline_cnt = 0;
/* main program */
    while ((read_char = getchar( )) != EOF)
        if (read_char = = ' ')
                blank_cnt = blank_cnt + 1;
        else if (read_char = = '\t')
                tab_cnt = tab_cnt + 1;
        else if (read_char = = '\n')
                newline_cnt = newline_cnt + 1;
    printf ("Number of newline character is %d\n", newline_cnt);
    printf ("Number of blank character is %d\n", blank_cnt);
    printf ("Number of tab character is %d\n", tab_cnt);
}
```

ב. יש לבצע את תרגיל 10-1 שבספר:

לכתוב תכנית המבצעת העתקת הקלט לפלט, תוך כדי החלפת כל טאב ב-\t, כל backspace ב-\b וכל backslash ב-\\. (כך הופכים מקשים אלו לנראים.)

פתרון

```
#include <stdio.h>
main ( )
{
    int read_char;  /* The char value we get from the input */
    while ((read_char = getchar( )) != EOF)
        if (read_char = = '\b')
                printf ("\ \b");
        else if (read_char = = '\t')
                printf ("\ \t");
        else if (read_char = = '\ \')
                printf ("\ \ \ \");
        else
                putchar(read_char);          /* usual case */
}
```

1.5.4 Word Counting

תרגיל 7

יש לבצע את תרגיל 12-1:

לכתוב תכנית המדפיסה את הקלט כאשר בכל שורה מופיעה מילה אחת. נניח שהתווים '\n', ' ', '\t' מציינים סוף מילה.

פתרון

```
#include <stdio.h>
#define IN 1
#define OUT 0
main ( )
{
    int read_char;
    int state;          /* Are we in the middle of a word? */
    state = OUT;    /* the beginning : not in the middle of a word */
    while ((read_char = getchar( )) != EOF)
        if (read_char = = ' ' | | read_char = = '\n' | | read_char = = '\t')
/* word separator encountered */
        {
           if (state = = IN)
/* if we were in the middle of a word and found a white space next to it */
                putchar ('\n');
/* then we print a newline */
           state = OUT;
/* in any case we are not in the middle from now on */
        }
        else
        {
                putchar(read_char);
                state = IN;
        }
}
```

1.6 Arrays

ההגדרה int ndigit[10] מגדירה מערך ששמו ndigit בעל עשרה איברים: ndigit[0] עד ndigit[9]. כל איבר במערך הוא מטיפוס int. לעיתים שוכחים שבהצהרה זו לא קיים האיבר ndigit[10]. בשפת C לא קיים range checking (בדיקת תחום) עבור מערך, ולכן פנייה ל-ndigit[10] לא תגרום שגיאה בזמן הידור. הסיבה לכך תוברר בפרק 5 העוסק במערכים. שם גם נציג דוגמאות לתופעות המתרחשות כאשר מבצעים גלישה כזאת. נדגיש כי דבר זה מהווה שגיאה נפוצה בשפה.

ההגדרה float num[5] מגדירה מערך בן חמישה איברים בשם num. ההתייחסות אל האיברים נעשית באמצעות הפניות num[0] עד num[4]. טיפוס כל איבר הוא float.

1.7 Functions

כאשר נדע להשתמש במשתני ה-shell של UNIX (לאחר קריאת יחידת הלימוד: "מדריך לסביבת UNIX"), נראה כי הערך שמחזירה הפונקציה main נכנס אל משתנה מיוחד. כך אפשר לבדוק בכל עת מהו הערך שהחזירה התכנית האחרונה שהורצה.

1.9 Character Arrays

תרגיל 8

יש להריץ את התכנית, המדפיסה את השורה הארוכה ביותר בקלט, שבעמוד 29, ולבחון את פעולתה .

נשים לב ללולאת ה-for בפונקציה getline, ובעיקר לתנאי היציאה מן הלולאה. התנאי מורכב משלושה חלקים: הראשון מוודא שאין חריגה מגבולות המערך, השני בודק אם התו שנקרא כרגע הינו התו סוף- הקובץ, והשלישי בודק אם התו הזה הינו סוף-שורה. מופיעות כאן שתי תכונות המאפינות את שפת C:

א. ניתן לקרוא ממש את התו EOF .

ב. הואיל והתנאים נבדקים תמיד **משמאל לימין לפי סדר הופעתם בשורה**, מובטח לנו שקודם ייקרא התו ורק אחר כך תיערך בדיקה אם זהו התו סוף-שורה.

תכונה נוספת המודגמת כאן: כל משפט השמה ב-C (לבד מעצם ביצוע ההשמה) מחזיר את אגף ימין של ההשמה. בערך החוזר ניתן להשתמש במשפט אחר. לדוגמה, משפט ההשמה

(to[i] = from[i])

מחזיר את הערך from[i], ובערך זה משתמשים במשפט ה-while. הערך החוזר מושווה לתו המציין סוף מחרוזת, '\0'. תכונה זו מאפשרת לכתוב השמה ותנאי של לולאת while במשפט אחד.

ניתן למשל לכתוב גם:

value = 7 * (to[i] = from[i])

הסוגריים דואגים לכך שמשפט ההשמה הימני יבוצע קודם לכפל. לפיכך הערך שהמשתנה value יקבל יהיה 7*from[i] . זהו קיצור של שני המשפטים:

to[i] = from[i]
value = 7 * from[i]

תרגיל 9

יש לבצע את תרגיל 1-19 שבספר: כתיבת פונקציה reverse(s) , אשר הופכת את התווים במחרוזת s. יש להשתמש בפונקציה כדי לכתוב תכנית ההופכת את הקלט שלה על ידי היפוך שורה בכל פעם.

פתרון

```
#include <stdio.h>

void reverse (char s[ ] )
/* reverse character string */
{
   int string_length;
   int i;
   char tmp;
   /* step 1: calculate length of s */
```

```
    string_length = 0;
    while ( s[string_length] != '\0')
        string_length = string_length + 1;
    /* step 2: reverse string */
    for (i = 0; i < string_length/2; i++)
    {
        tmp = s[i];
        s[i] = s[string_length – i – 1];
        s[string_length – i – 1] = tmp;
    }
}
main ( )
{
    char string[50];  // max length is 50
    string[0] = 'h';
    string[1] = '0';
    string[2] = 'w';
    string[3] = '  ';
    string[4] = 'a';
    string[5] = 'r';
    string[6] = 'e';
    string[7] = '  ';
    string[8] = 'y';
    string[9] = 'o';
    string[10] = 'u';
    string[11] = '\0';                    /* Mark end of string */
    printf ("before calling reverse string is %s\n", string);
    reverse(string);
    printf ("after calling reverse string is %s\n", string);
}
```

נקודות לסיכום פרק 1

פרק זה מהווה תקציר מהיר לשפה. הנושאים העיקריים שנדונו בו הם:

- כיצד לכתוב תכנית קצרה.
- מהי פונקציית ספרייה: הספרייה הסטנדרטית אינה חלק משפת C.
- שימוש במשתנים מסוגים שונים ובביטויים המכילים משתנים שונים.
- כיצד להדפיס משתנים שונים.
- כיצד להגדיר קבועים על ידי #define.
- כיצד מקדדים תוים, כיצד נקרא תו.
- מהו מערך בשפת C.

תזכורות שהופיעו בפרק:

- שיטת המשלים ל-1 (one's complement).
- שיטת המשלים ל-2 (two's complement).

פרק 2. Types, Operators and Expressions

2.1 Variable Names

שם בלתי חוקי יזוהה על ידי המהדר. דוגמאות לשמות משתנים בלתי חוקיים:

34size – מתחיל במספר

dollar$ – מכיל תו אסור

float – שם שמור

שם כגון pkpk הוא חוקי, אולם אינו משמעותי. יש להשתמש בשמות משמעותיים, לשם הבנת הקוד ותחזוקו. מובן שבנוסף יש לתעד ולהסביר את תפקידו של כל משתנה בתכנית. בתכניות גדולות ניתן להשתמש במעין מילון, ממוין לפי א-ב, לשם תיעוד המשתנים.

2.2 Data Types and Sizes

בשפת C ניתן לקבוע מראש, באמצעות המציינים (qualifiers) long ו-short, מהו גודל הזיכרון אשר יוקצב למשתנה מסוים. מציינים אלה רשאים להופיע לפני כל אחד מארבעת טיפוסי המשתנים הבסיסיים של השפה:

char, int, float, double

המציינים הם אופציונליים, אולם הסדר שבו הם מופיעים הוא חשוב. תחילה מופיעה מילת הסימן, signed או unsigned, אחריה הגודל, short או long, ואחריו הטיפוס הבסיסי עצמו שהוא אחד מבין הארבעה הנ"ל.

לדוגמה:

```
unsigned short int  out_flag;
long int  sum;
```

ניתן למצוא מהו גודל הזיכרון אשר תופס כל אחד מטיפוסי הנתונים הנ"ל, יחסית לגודלו של טיפוס char, באמצעות האופרטור sizeof אשר יילמד בהמשך.

האפשרות לקבוע את גודל הזיכרון שיוקצה למשתנה מהווה מרכיב של אחת התכונות הייחודיות של שפת C: היכולת לשלוט במכונה באמצעות שפה עילית. תכונה זו נתמכת גם על ידי היכולת לקבוע אם משתנה שלם יהיה בעל סימן (signed) או חסר סימן (unsigned) במסגרת ביצוע פעולות אריתמטיות.

ב-Appendix-B בעמוד 257 בספר מופיע תיאור קובץ ה-header הסטנדרטי <limits.h>. בקובץ זה מופיעה רשימת קבועים סימבוליים אשר חייבים להיות מוגדרים עבור כל מהדר התומך ב-ANSI-C. הערכים של הקבועים הסימבוליים המתוארים בספר הם ערכי מינימום. ייתכן שהמהדר שבו משתמשים מגדיר ערכים אחרים עבור הקבועים הסימבוליים, אולם שמותיהם חייבים להיות זהים לאלה שמופיעים ב-Appendix-B.

מומלץ להתבונן בקובץ <limits.h>, ולבדוק מה הם ערכי הקבועים הסימבוליים (אשר מוגדרים ב-<limits.h>). מתוך ערכים אלה אפשר לדעת מה הם גבולות המהדר מבחינת חישובים אריתמטים.

תרגיל 10

בתרגיל 2-1 בספר נדרש לכתוב תכנית הקובעת את התחום של char, short, int, long הן עבור signed והן עבור unsigned.

יש לפתור את תרגיל 2-1 (קביעת התחום של טיפוסי הנתונים) תוך כדי שימוש בערכי הקבועים הסימבוליים המופיעים ב-<limits.h>. יש לקבוע את התחום עבור long, int, short, char. עבור כל טיפוס, יש למצוא את התחום עבור המציינים signed ו-unsigned. (%u מודיע ל-printf כי ברצוננו לקבל הדפסה של %ld unsigned int משמש להדפסת long.) ניתן לעיין בפתרון הבא:

פתרון

```
#include <stdio.h>
#include <limits.h>
main ( )
{
    printf ("The range of signed char is from %d to %d\n",
                SCHAR_MIN, SCHAR_MAX);
    printf ("The range of unsigned char is from %d to %d\n",
                0, UCHAR_MAX);
    printf ("The range of int is from %d to %d\n",
                INT_MIN, INT_MAX);
    printf ("The range of unsigned int is from %d to %u\n",
                0, UINT_MAX);
    printf ("The range of short is from %d to %d\n",
                SHRT_MIN, SHRT_MAX);
    printf ("The range of unsigned short is from %d to %u\n",
                0, USHRT_MAX);
    printf ("The range of long is from %ld to %ld\n",
                LONG_MIN, LONG_MAX);
    printf ("The range of unsigned long is from %d to %lu\n",
                0, ULONG_MAX);
}
```

Constants 2.3

תזכורת : בסיס הקסדצימלי

בסיס הקסדצימלי הוא בסיס 16, כלומר מספר הספרות בבסיס הוא 16. משתמשים בספרות 0-9 ולצורך 6 הספרות הנוספות משתמשים באותיות אנגליות:

a = 10	A = 10
b = 11	B = 11
c = 12	C = 12
d = 13	D = 13
e = 14	E = 14
f = 15	F = 15

דוגמה

נתרגם את המספר העשרוני 2649:

$2649 = A\times16^2 + 5\times16^1 + 9\times16^0 = A59_{16}$

ובבסיס בינארי:

$2649 = 101001011001_2$

נמקם את המספרים זה מעל זה:

A 5 9
1010 0101 1001

נשים לב שכל ארבע ספרות בינאריות מתאימות לספרה ההקסדצימלית שמעליה. תופעה זו מתקיימת תמיד. לפיכך מתייחסים אל מספר הקסדצימלי כאל "קיצור" של מספר בינארי, כאשר כל ספרה ניתנת לפירוק לארבע ספרות בינאריות. מכיוון שבמחשב עובדים עם יחידות בסיסיות של 8 סיביות (ביטים), נוח לייצג ספרות באמצעות הבסיס ההקסדצימלי.

בשפת C , כאשר רוצים לציין מספר בבסיס הקסדצימלי, כותבים 0x אם האותיות ההקסדצימליות שבהן משתמשים הן a-f, ו-0X אם האותיות ההקסדצימליות שבהן משתמשים הן A-F.

מדוע יש צורך באותיות U ו-L ליד קבועים? נשיב בהרחבה על שאלה זו. נניח שהגדרנו פונקציה בשם func המקבלת פרמטר אחד מטיפוס unsigned long int.
בהפעלה מסוימת של הפונקציה אנו מעוניינים לקרוא לה עם הערך הקבוע 0xf (הערך 15). כאשר המהדר מזהה את הקבוע 0xf, הוא מחליט **בעקיפין** (implicitly) שזהו קבוע מטיפוס signed int, אלא אם כן נגדיר במפורש שזהו טיפוס unsigned long int.

מדוע דווקא int? ההחלטה אם קבוע הוא int או long היא פשוטה: אם ניתן לאחסן את הקבוע בגודל של int, אזי הוא מטיפוס int, אחרת הוא מטיפוס long (בהנחה שניתן לאחסנו ב-long, ואם לא– תתקבל שגיאה בזמן ההידור). כעת, מכיוון שאת 15 ניתן לייצג על ידי 16 סיביות (שהוא האורך המובטח לטיפוס int), המהדר יחליט שלפניו מספר מטיפוס int.

מדוע signed? הסיבה לכך היא ברירת המחדל לגבי קבועים, המניחה שהם בעלי סימן.

הפונקציה func מצפה, כאמור, לפרמטר שגודלו ארבעה בתים (הגודל המובטח ל-long). אולם, מכיוון שהמהדר החליט שהפרמטר הקבוע הוא מטיפוס int, הוא מעביר שני בתים בלבד. אבל func אמורה לקרוא מהזיכרון ארבעה בתים, ולכן ייקראו שני הבתים הנוספים מהמשך המחסנית. הערך שייקרא מהזיכרון לא יהיה כמובן 0xf המיועד, והוא יוביל לתוצאה שגויה.
הדרך להתגבר על הבעיה היא לקרוא ל-func בצורה:

```
func ( 0xfUL );
```

בכך מבטיחים שיועבר אל הפונקציה שלם מטיפוס long.

אם ידוע למהדר מהו הטיפוס שלו מצפה הפונקציה func, אזי הוא יתרגם בעקיפין את הקבוע לטיפוס המתאים, שלו מצפה הפונקציה. אולם, אם טיפוסי הפרמטרים לא ידועים למהדר (למשל, כאשר func היא פונקציית ספרייה הנטענת לתוך הקוד באמצעות ה-linker/loader), ההחלטה מה להעביר תבוצע כפי שתואר לעיל, והיא עלולה לגרום לשגיאה בזמן הריצה.

מסיבה זו כדאי לתת למהדר מידע על טיפוסי פרמטרים של פונקציות. ANSI-C מספקת את האפשרות לעשות זאת באמצעות **אבות-טיפוס** (prototypes). אב-הטיפוס

מודיע למהדר מה טיפוסי הפרמטרים שלהם מצפה הפונקציה (אפשרות שלא הייתה קיימת במהדורה הראשונה של השפה). כאשר יגלה המהדר חוסר התאמה בטיפוסים, הוא יתריע על כך.

לדוגמה, בפונקציה func, נשתמש באב- טיפוס:

```
func( unsigned long int x)
```

נשים לב כי שימוש באב- טיפוס מקל גם על הבנת התכנית ותיעודה.

2.4 Declarations

נזכיר כי משתנה אוטומטי ב-C הוא משתנה המוגדר כמקומי (local) לפונקציה. (קטע 1.10 בספר, עמוד 31). נשים לב כי משתנה זה אינו מוכר מחוץ לפונקציה. אם שם זה מופיע מחוץ לפונקציה – מדובר במשתנה אחר, כלומר תיתכן חפיפת שמות.

דוגמה

כאשר, למשל, יוגדר משתנה בשם x, הן בפונקציה והן מחוצה לה, כמשתנה גלובלי. ההתייחסות ל-x בתוך הפונקציה תהיה למשתנה המקומי, ומחוץ לפונקציה ההתייחסות ל-x תשויך למשתנה הגלובלי. מובן שדבר זה אינו מומלץ, והוא מקור לבלבול ולטעויות.

בדוגמת הקוד שלהלן, ערך המשתנה הגלובלי x יהיה 5, בעוד שערך המשתנה המקומי x יהיה 3.

```
#include <stdio.h>
int x=5;
…
void func (void)
{
   int x;
   x=3;
   .
   .
   .
}
```

Arithmetic Operator 2.5

יש חשיבות לסדר הפעולות. סיכום לסדר זה מופיע בטבלה שבעמוד 53 בספר. תמיד עדיף להשתמש בסוגריים לשם בהירות.

Relational and Logical Operators 2.6

הביטוי '\n'=!(c=getchar( )) משתמש בתכונה שמשפט השמה מחזיר ערך. הערך של חלקו הימני של המשפט c=getchar( ) (כלומר הערך שמחזירה הפונקציה getchar( )), מושווה כנגד התו '\n'.

ה-and הלוגי, &&, וה-or הלוגי, ||, הם בעלי עדיפות נמוכה מזו של אופרטורי היחס, ולכן אין צורך להקיף בסוגריים את ביטויי היחס.

יש לשים לב שביטויי היחס והביטויים הלוגיים מחזירים אף הם ערך. ערך זה הוא 1 או 0, בהתאם לנכונות הביטוי (כלומר, אם הביטוי הוא TRUE, יוחזר הערך 1). הערך החוזר הוא **שלם**. ההצהרה והאתחול:

```
int x = (49 == 7*7)
```

תציב ב-x את הערך 1. הסוגריים למעשה מיותרים (הסיבה לכך תובהר בסוף הפרק, כאשר נלמד על סדר העדיפות של האופרטורים). נדגיש שוב: למרות שהם מיותרים, חשוב להשתמש בסוגריים לשם בהירות.

Type Conversions 2.7

יש לשים לב: קובץ ה-header <ctype.h> אינו **מגדיר** (defines) את הפונקציות כפי שטוען הספר, אלא רק **מצהיר** (declares) עליהן (הפסקה הרביעית בעמוד 43). על ההבדלים שבין הצהרה להגדרה נעמוד בפרק 4 העוסק בפונקציות. ההגדרה – קוד הפונקציות (המוצהרות ב-<ctype.h>) – נמצאת בספרייה הסטנדרטית של C (libc.a) לדוגמה), ומצטרפת לקוד בזמן הקישור (linking). רשימת הפונקציות המוצהרות ב-<ctype.h> מתוארת ב-Appendix-B בעמוד 248. (אם נתבונן בקובץ ctype.h, נגלה

שרוב הפונקציות האלה הן למעשה macros שה-C pre-processor דואג לשתול בקוד לפני ההידור).

לגבי האופן שבו מבוצעת המרה (conversion) של שלם מטיפוס char לשלם מטיפוס int או long, ניתן לקבוע באיזו משתי הצורות אנו מעוניינים:

א. char הוא שלם בעל סימן (זוהי ברירת המחדל - default).
ב. char הוא שלם חסר סימן.

הדבר נעשה עבור המהדר בעזרת אופציות חיצוניות.

לדוגמה, אם כתבנו:

```
char x = '\xff';
```

ואחר כך:

```
printf("%d\n",x);
```

הרי במקרה הראשון (של ה-default) יודפס 1–, ובשני 255.

בהגדרה התקנית לא נקבע בכוונה תחילה אם ברירת המחדל של טיפוס הנתונים char היא signed או unsigned. זאת מכיוון שבמחשבים מסוימים signed chars הינם יעילים יותר מאשר unsigned chars, ובמחשבים אחרים ההפך הוא הנכון.

ההגדרה התקנית מאפשרת לכותב המהדר לבחור מה תהיה ברירת המחדל של char, ומתיר למשתמש לציין במפורש את הטיפוס שבו הוא מעוניין אם הוא שונה מברירת המחדל שהמהדר מספק.

cast הוא **אופרטור** אונארי. חשוב לזכור זאת. כמו לכל אופרטור, יש לו עדיפות שהיא יחסית לאופרטורים אחרים (כפי שמופיע בסוף הפרק, בטבלה בעמוד 53).

הבהרה לגבי תרגילים 12, 11 שבהמשך:
תוצאות התרגילים יכולות להיות שורש של 2, למרות שחסרה הגדרת אב הטיפוס של הפונקציה sqrt. אולם הדבר אינו מובטח (תלוי בסביבה). במקום sqrt אפשר להשתמש בפונקציה חדשה אשר נמצאת בקובץ אחר. למשל:

```
double my_div(double num)
{
        return num/2;
}
```

במקרה זה אכן לא נקבל שורש 2.

תרגיל 11

יש לכתוב ולהריץ את קטע התכנית הבא (חישוב והדפסת שורש 2):

```
#include <stdio.h>
main( )
}
   printf ("%f\n", sqrt(2))
}
```

יש להתעלם מהודעת האזהרה של המהדר תוך כדי ההידור (ל-sqrt לא הוצהר אב-טיפוס).
התוצאות, כפי שרואים, בהסתייגות הבהרה בסוף הסעיף, אינן שורש 2. מדוע? התשובה פשוטה: לא הוצהר איזה טיפוס מחזירה הפונקציה sqrt. ברירת המחדל ב-C היא שהטיפוס המוחזר הוא int (כפי שכתוב בפסקה הראשונה בעמוד 27 ובהגדרת השפה Appendix-A, קטע 10.1). הפונקציה printf ציפתה למספר ממשי (בגלל %f) וניסתה לקודד את השלם שהחזירה sqrt כייצוג של מספר ממשי, ולכן קיבלנו "זבל".

ננסה אפוא לתקן זאת, על ידי הצהרה על sqrt כפונקציה המחזירה double.

```
#include <stdio.h>
main( )
{
   double sqrt( );
}
```

שימו לב שלא הצהרנו כאן מהו טיפוס הפרמטר ש-sqrt מקבלת, אלא רק איזה טיפוס היא מחזירה. זוהי צורת ההצהרה שהייתה נהוגה ב-C לפני שהוצע תקן ANSI.
נריץ את התכנית המתוקנת ולאכזבתנו נקבל שוב תוצאה לא נכונה.

מדוע? הפעם גרמה לכך העובדה שהפונקציה sqrt מצפה ל-double. אולם, כאשר עובר המהדר על הקוד, אין הוא יודע לאיזה טיפוס מצפה sqrt (sqrt היא פונקציית ספרייה, וככזאת היא כבר קודדה לשפת מכונה). היות ש-2 הוא מטיפוס int, המהדר יוצר קוד שבו מועבר המספר השלם 2. אולם sqrt מצפה ל-double, ומנסה לפרש את השלם 2 כקידוד של מספר ממשי. אך כידוע הקידוד של 2 שונה מקידוד של 2.0, ולכן התוצאות אינן נכונות (נזכור את הדוגמה של func לעיל).

ישנן שתי דרכים להתגבר על הבעיה. הראשונה (כפי שנעשה הדבר לפני ANSI-C), היא לדאוג מפורשות שאכן יועבר מספר ממשי ל-sqrt. ניתן לבצע זאת בשתי שיטות:

בשיטה הראשונה נכתוב 2.0 בצורה מפורשת.

```
include <stdio.h>
main( )
{
   double sqrt( );
   printf ("%f\n", sqrt(2.0));
}
```

שיטה זו אינה שימושית כאשר מעבירים משתנה.

השיטה השנייה משתמשת באופרטור cast.

```
#include <stdio.h>
main( )
{
   double sqrt( );
   printf("%f\n", sqrt(double 2));
}
```

שיטה זו מתבססת על אבות-טיפוס ב-prototypes של פונקציות. זהו החידוש העיקרי שהכניסה הוועדה. אנו מצהירים כאן לא רק על טיפוס הפונקציה, אלא גם על טיפוסי הפרמטרים. השיטה תעבוד כמובן רק במהדרים התומכים ב-ANSI-C (כיום כמעט כל המהדרים תומכים בתקן זה).

גרסה אחרת לשיטה השנייה:

```
#include <stdio.h>
main((
{
   double sqrt(double );
   printf("%f\n", sqrt(2));
}
```

בגרסה זו, כאשר המהדר נתקל לראשונה ב-2 הוא כבר יודע ש-sqrt מצפה ל-double (הודות להצהרה על אב-הטיפוס), ולכן מקודד את 2 כמספר ממשי ולא כשלם. למעשה נעשה כאן cast בצורה סתומה (implicitly), ולא בצורה מפורשת כמו בגרסה הקודמת.

קובץ ה-header הסטנדרטי <math.h> מכיל את אבות-הטיפוס של כל הפונקציות המתמטיות הנמצאות בספרייה של C. בין השאר מופיעה שם השורה

double sqrt (double)

ולכן די לכתוב בתחילת הקובץ:

include <math.h>

ואין למעשה צורך להצהיר במפורש על sqrt כמו בדוגמה לעיל.

תרגיל 12

לאור הנאמר לעיל, ננסה לחזות מראש אם שתי התכניות הבאות תדפסנה את שורש 2 כיאות ומדוע.

```
#include <stdio.h>
main( )
{
   printf ("%f\n", sqrt(2.0));
}
```

```
#include <stdio.h>
main( )
{
   printf ("%f\n", (double) sqrt(2.0));
}
```

פתרון

שתי התכניות מובילות לתוצאות שגויות. אמנם מועבר בשתיהן double, אך מכיוון שלא הוצהר מהו הטיפוס ש-sqrt מחזירה, ברירת המחדל להנחת המהדר היא int. לכן יפרש המהדר את הערך שהוחזר כ-int, והתשובה תהיה אפוא שגויה. ה-cast המופיע בתכנית השנייה פועל על הערך המוחזר עצמו (כי לפונקציה יש עדיפות על cast). מנקודת הראות של המהדר, ה-cast פועל על ה-int שהפונקציה החזירה.

הבהרה:

התשובה השגויה מתקבלת, בתלות בקומפיילר, כך שלא תמיד נקבל תשובה שגויה, ולעיתים נקבל דווקא את התשובה הנכונה (שורש 2).

אם בקומפיילר שלנו גודלו של int הוא 4 בתים בדרך-כלל, אז ניתן לקלוט את רוב הנתונים של משתנה מסוג double.

כך שאם פונקציה מוצהרת כ-int, בטעות (בגלל קריאה לפונקציה, ללא הצהרה), אז ההמרה ל-4 בתים מאבדת רק את הדיוק של הספרות אחרי הנקודה.

לכן כדאי, כתרגיל, לבדוק ולהדפיס את הדיוק המתקבל ב-4 בתים ואת הדיוק ב-double, ולבחון את מספר הספרות המשמעותיות.

Increment and Decrement Operators 2.8

תרגיל 13

מדוע לא נכתבה לולאת ה-while הראשונה בפונקציה strcat (עמוד 48) בצורה:

```
while (s[i++] != '\0')
;
```

פתרון

אם תתבצע הלולאה כך, נצטרך להפחית בסופה 1 מ-i, על מנת של-i יהיה הערך המתאים בכניסה ללולאה השנייה.

השימוש באופרטורים ++ ו -- דורשים מתן תשומת לב מיוחדת מצד המתכנתים. שהרי משפט כגון

```
i = 5;
s[i] = i++;
```

הינו דו-משמעי. הערך שיושם לתוך s[i] הוא 5. אולם C לא קובעת אם הקידום של i יבוצע לפני או אחרי חישובו של i באגף שמאל של ההשמה. ייתכן שהערך שישונה הוא s[5], אך ייתכן גם שיהיה זה s[6].

תרגיל 14

יש לבצע את תרגיל 4-2:

יש לכתוב גרסה נוספת לפונקציה squeeze(s1, s2), אשר מוחקת כל תו ב-s1, אשר מתאים לתו במחרוזת s2.

פתרון (4-2)

```
/* Improved squeeze function. The main idea is to apply the function (book page 47)  on each character of s2*/
#include <stdio.h>
void squeeze_char (char s[ ], int c)
/* This is the squeeze function (the book page 47) */
{
   int i, j;
   for (i = j = 0; s[ i ] != '\0'; i++)
      if (s[ i ] != c)
            s[ j++] = s[ i ];
   s[ j ] = '\0';
}
void squeeze (char s1[ ], char s2[ ])
/* This is the squeezing function. It calls squeeze_char for each character of s2 */
{
   int i;
```

```
    for (i = 0; s2[ i ] != '\0'; i++)
        squeeze_char (s1, s2[ i ]);
}
main ( )
{
    char string1[40] = "hello";
    char string2[40] = "world";
/* The two lines above allocate storage of 40 bytes for each string. The assignment
here acts as an initialization. It is equivalent to writing
    char string1[40];
    string1[0] = 'h';
    string1[1] = 'e';
        .
        .
    string1[5] = '\0';
you cannot assign a string to a character array by the assignment operator. You can
only do it in the initialization. */
    printf ("before squeeze, string1=%s and string2=%s\n", string1, string2);
    squeeze (string1, string2);
    printf ("after squeeze, string1=%s and string2=%s\n", string1, string2);
}
```

Bitwise Operators 2.9

תרגיל 15

יש לבצע את תרגיל 2-8: יש לכתוב פונקציה בשם rightrot(x, n) , אשר מחזירה את ערכו של integer x, לאחר סיבוב לימין (רוטציה) של n מקומות.

פתרון (2-8)

```
include <stdio.h>
/* The procedure is simple. First we calculate the number of bits in as
integer. Then we make a right shift to x by n, then a left shift of (original) x
by (length of int – n) and do the 'or' operator on the above two results */
```

```
int int_len(void)
{
    unsigned int x = ~0;                    /* x is full of ones */
/* if you don't understand why we used unsigned integer for x, read the
second paragraph of page 49 */
    int i;
    for (i = 0; x != 0; i++)
        x = x >> 1;
    return ( i );
}
int reghtrot (unsigned int x, int n)
{
    return ( (x >> n) | (x << (int_len( ) – n)) );
}
main( )
{
    int i = 0xfff0;
    printf ("%x\n", rightrot(i, 3));
/* Note the implicit cast of i, due to rightrot prototype */
}
```

Assignment Operators and Expressions 2.10

תרגיל 16

יש לבצע את תרגיל 2-9:

בשיטת המשלים ל-2, x &= (x-1) מוחק את ה-1-ים הימניים ביותר ב-x. יש להסביר מדוע, ולהשתמש בהסבר זה, כדי לכתוב גרסה מהירה יותר של הפונקציה bitcount.

פתרון (2-9)

הסבר:

אם x הוא אי -זוגי, אזי ייצוגו של (x–1) זהה, פרט לכך שהביט הימני ביותר הוא עכשיו 0 (במקום 1). במקרה זה (x–1) == (x & (x–1)) .

אם x הוא זוגי, אזי בייצוגו של (x–1) ה-0-ים הימניים ביותר הופכים ל-1-ים, וה-1-ים הופכים ל-0-ים. ה-And בין שניהם מאפס את ה-1-ים הימניים ביותר ב-x ואת כל ה-1-ים הימניים ביותר ב-(x–1).

בעמוד 50 בספר כתובה הפונקציה bitcount באופן הבא:

```
/* bitcount:  count 1 bits in x */
int bitcount(unsigned x)
{
   int b;

   for (b = 0; x != 0; x >>= 1)
      if (x & 01)
         b++;
   return b;
}
```

לאור ההסבר למעלה, נכתוב את הגרסה המהירה לפונקציה bitcount:

```
/* bitcount:  count 1 bits in x */
int bitcount(unsigned x)
{
   int b;

   for (b = 0; x != 0; x &= (x-1))
      b++;
   return b;
}
```

Conditional Expressions 2.11

תרגיל 17

יש לבצע את תרגיל 10-2:

יש לכתוב מחדש את הפונקציה lower, אשר ממירה אותיות גדולות (upper case) לאותיות קטנות (lower case), עם ביטוי שהוא תנאי (במקום שימוש ב- if-else).

פתרון (2-10)

```
#include <stdio.h>
/* Conditional lower */
/* ASCII only */
char lower (char c)
{
    return ( (c >= 'A' && c <= 'Z') ? c + 'a' – 'A' : c);
}
main( )
{
    printf ("%c\n", lower ('F'));
    printf ("%c\n", lower ('f'));
}
```

ל-C מגוון אופרטורים רחב. חלקם הגדול תואר בפרק זה (סיכום האופרטורים מופיע בטבלה, בעמוד 53).

נקודות לסיכום פרק 2

הנושאים העיקריים שנדונו בפרק זה הם:

- שמות חוקיים למשתנים
- גודל הזיכרון וסוגי משתנים
- קבועים בשפת C
- אופרטורים

תזכורות שהופיעו בפרק:

- בסיס הקסדצימלי

פרק 3. Control Flow

3.1 Statements and Blocks

לכתיבת הסוגריים המסולסלים מקובלות שתי גישות של התאמה (allignment): באחת (המודגמת בתכניות שבספר), הסוגר הפותח לא נכתב בשורה נפרדת אלא בהמשך שורת ה-statement הכוללת את המשפט המורכב. השנייה (המודגמת בפתרונות המובאים כאן) מייחדת לסוגר הפותח שורה נפרדת. יש לבחור אחת משתי הגישות, ולהישאר נאמן לה.

3.2 If-Else

שגיאת תכנות נפוצה היא לרשום משפט if בסגנון:

```
if (a = 0)
    statement;
```

כאשר רוצים לבדוק אם המשתנה a **שווה** ל-0.
כפי שראינו, משפט השמה מחזיר ערך. ערך זה נבדק במשפט ה-if. במקרה זה, המשפט (a = 0) יחזיר את הערך 0, אשר נחשב ב-C כ-false, ולכן ה-statement לא יבוצע לעולם.
אין בצורת הכתיבה הנ"ל משום שגיאה תחבירית או סמנטית, ולכן לא כל מהדר יתריע על כך, **למרות שברור** כי המשפט לעולם לא יבוצע (תרגיל: בדקו זאת לגבי המהדר שבו אנו משתמשים).
הדרך הנכונה לכתוב היא, כמובן:

```
if (a == 0)
    statement;
```

Switch 3.4

ב-falls through, הכוונה בהסבר שבפסקה הראשונה בעמוד 59 היא שכל עוד לא נתקלים בהוראת break או בתו '}' המסיים את משפט ה-switch, הפקודות ימשיכו להתבצע. אילו בתכנית שבעמוד 59 היינו משמיטים את ה-break הראשון, אזי כאשר התו c נמצא בין '0' ל '9' ('0'<=c<='9') , היו מתבצעים שני המשפטים הבאים:

```
ndigit[c–'0']++;
nwhite++;
```

אין אפשרות להגדיר תחום במשפט switch, כלומר לשם בדיקה אם התו ch נמצא בתחום, תבוצע ב-C בדיקה כנגד **כל** ערך המופיע במשפט ה-switch.
אם מספר הערכים הוא רב, וניתן לתחום את ערכי הבורר (ch במקרה הנ"ל), עדיף להשתמש במשפט if ולא במשפט switch.

Loops-While and For 3.5

שימו לב שלולאת for של שפת C חזקה יותר מלולאת for של שפות אחרות. אין היא חייבת להיות קשורה במשתנה לולאה (control variable). למעשה, היא מתורגמת למשפט while , כפי שמתואר בפסקה הראשונה של הסעיף.

לעיתים אין אנו מעוניינים כלל ב-statement של לולאת for אלא רק בתוצאות הלוואי של הלולאה. למשל, אם נרצה לדעת היכן נמצא (כאינדקס) התו סוף-מחרוזת '\0' במערך התווים s, נוכל להסתפק בלולאה כגון:

```
for (i = 0; s[ i ] != '\0'; i++)
    ;
```

שימו לב שבין התו ";" לבין ביטוי ה-for אין כל הוראת ביצוע. זהו משפט ביצוע ריק. אנו מעוניינים בערך i בעת היציאה מן הלולאה. כאשר מבצעים משפט ריק, כדאי להדגיש זאת באמצעות כתיבת ";" בשורה נפרדת, כפי שמודגם בלולאה המדלגת על רווחים בעמוד 61:

```
for (i = 0; isspace(s[i]); i++)
    ;
```

שגיאת תכנות נפוצה היא לרשום ; לאחר הסוגריים של משפט ה-for, דבר המונע את ביצוע המשפט שבו אנו מעוניינים בתוך הלולאה:

דוגמה לקוד **שגוי**:

```
for (i = 0; i < N; i++);
    printf("hello \n");
```

כאן נצפה ל-N הדפסות של המילה hello, כאשר בפועל נקבל הדפסה אחת בלבד.

התיקון:

```
for (i = 0; i < N; i++)
    printf("hello \n");
```

אופרטור הפסיק (,) הוא **אופרטור בינארי**. כמו כל אופרטור בינארי (כפל, חיבור, חיסור וכו'), הוא מחייב אופרנדים בשני צדדיו. על האופרנדים להיות ביטויים (expressions) בשפה. הערך שהאופרטור מחזיר הוא ערכו של הביטוי הימני. סדר הביצוע של הביטויים הוא משמאל לימין.

הוויכוח סביב השאלה מתי צריך להשתמש באופרטור הזה ומתי לא – ימיו כימי השפה עצמה. בדרך כלל משתמשים בו רק בביטויים הקשורים ביניהם מבחינה רעיונית, בעיקר בלולאות for. שם אין משתמשים בערך האופרטור, אלא הוא משמש להפרדה בין צעדי האתחול, בחלק הראשון של משפט for ובין צעדי הקידום (increment) בחלק השלישי של המשפט, כפי שנראה בדוגמה שבעמוד 62:

```
for (i = 0, j = strlen(s)-1; i < j; i++, j--)
...
```

במקרים רבים אפשר להשתמש באופרטור הפסיק במקום ב-;. אך הדבר אינו מקובל, ומצביע על תכנות גרוע.

הפונקציה strlen היא פונקציית ספרייה סטנדרטית המחזירה את אורך המחרוזת. מחרוזת שהתו הראשון שלה הוא '\0' נקראת **מחרוזת ריקה** ("), ואורכה הוא 0.

תרגיל 17

יש לבצע את תרגיל 3-3. יש לכתוב פונקציה בשם expand(s1, s2) אשר מרחיבה קיצורים כגון a-z במחרוזת s1 לרשימה המלאה השקולה לכך: abc...xyz במחרוזת s2. יש לאפשר אותיות גדולות וקטנות, כמו גם ספרות.
אין צורך לטפל במקרים כגון a-z- . כלומר, נניח כי המחרוזת היא בעלת תחביר חוקי, ומופיעים בה ביטויים מהצורה a-z0-9b-s בלבד (ולא כפי שמתואר בשאלה).

פתרון (3-3)

```
#include <stdio.h>
expand (char expan_str[ ], char target [ ])
{
    int target_ndx = 0;
    int exp_ndx = 0;
    char from, to, mid;
    while (expan_str[exp_ndx] != '\0')
    {
        from = expan_str[exp_ndx++];
        mid = expan_str[exp_ndx++];
        to = expan_str[exp_ndx++];
        if (mid != '–')
                return (–1);                    /* Error indication */
        if (from > to)
        {
                /* Swap. Use mid  */
                mid = to;
                to = from;
                from = mid;
        }
        do
                target[target_ndx++] = from;
        while (from++ < to);
    }
    target[target_ndx] = '\0';
```

```
    return (0);                          /* Success indication */
}
main ( )
{
    char target[100];
    char expn[40] = "a–s0–9D–b";
    if (expand (expn, target) < 0)
        printf ("Error in expansion format\n");
    else
        printf("target after expansion is %s\n", target);
}
```

Loops - Do-While 3.6

תרגיל 18

יש לבצע את תרגיל 3-6. יש לכתוב גרסה לפונקציה itoa, אשר תקבל שלושה ארגומנטים במקום שניים (כפי שמופיע בעמוד 64). הארגומנט השלישי יהיה רוחב מינימלי של שדה. כלומר, ייתכן שיש להוסיף רווחים משמאל למספר המומר, כך שהרוחב יתאים למינימום הנדרש.

פתרון (3-6)

```
#include <string.h>
#include <stdio.h>
#include <limits.h>
#define BUFLEN 20
#define MINWIDTH 7

void itoa(int n, char s[], int width);
void reverse(char s[]);

int main(void) {
```

```
    char buffer[BUFLEN];

    itoa(INT_MIN, buffer, MINWIDTH);
    printf("Buffer:%s\n", buffer);

    return 0;
}

/* convert n to characters in s, with minimum field width */
void itoa(int n, char s[], int width) {
    int i, sign;

    if ((sign = n) < 0)  /* record sign */
        n = -1*n;  /* make n positive */
    i = 0;
    do {     /* generate digits in reverse order */
        s[i++] = n % 10 + '0';  /* get next digit */
        printf("%d %% %d + '0' = %d\n", n, 10, s[i-1]);
    } while ((n /= 10) > 0); /* delete it */
    if (sign < 0)
        s[i++] = '-';

    while (i < width )    /*  addition to original function  */
        s[i++] = ' ';

    s[i] = '\0';
    reverse(s);
}

/* reverse characters in s */
void reverse(char s[]) {
    int c, i, j;
    for ( i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i];
```

```
        s[i] = s[j];
        s[j] = c;
    }
}
```

Break and Continue 3.7

הפקודה break מאפשרת לצאת מלולאות (for, while) כאשר תנאי מסוים מתקיים באמצע הלולאה. היא חוסכת את הדגל הבוליאני אשר נבדק בתחילת הלולאה, ומונף כאשר התנאי אכן מתקיים. יש לציין שלא ניתן לצאת באמצעות break אלא מהלולאה הפנימית ביותר.
לדוגמה:

```
#define NUM 10
for (i = 0; i < NUM; i++)
        .
        .
        .
    while (j < NUM)
        break;
        .
        .
```

במקרה זה, הלולאה הראשונה תתבצע כרגיל (עד שערך המשתנה i יגיע ל-NUM), בעוד הפקודות בלולאת ה-while הפנימית לא יתבצעו כלל.

תרגיל 19

מה יקרה כאשר תתבצע ההוראה continue בתוך לולאת for המקוננת בלולאת for אחרת? לדוגמה:

```
for (i = 0; i < n; i++)
        .
        .
        .
        for (j = 0; j < n+7;  j++)
            .
            .
                continue
```

פתרון

הפקודה תגרום רק לביצועו של החלק השלישי במשפט ה-for הפנימי. כלומר בדוגמה יוגדל ערכו של j , ולאחר מכן ייבדק התנאי של הלולאה.

נקודות לסיכום פרק 3

הנושאים העיקריים שנדונו בפרק זה הם:

- בחירת סגנון אחיד למיקום סוגריים מסולסלים (ציון בלוק).
- מימוש לולאות בשפה.
- שיטות ליציאה מוקדמת מלולאה.

פרק 4. Functions and Program Structure

בהקדמה לפרק בספר (בשלוש הפסקאות האחרונות) מתוארים בקצרה ההבדלים העיקריים שבין הגרסה הקודמת של שפת C לבין ANSI-C. תיאור זה אינו רלוונטי למי שלא מכיר את הגרסה הראשונה, ואפשר לדלג עליו.

4.1 Basics of Functions

להלן כמה הסברים לגבי התכנית שבעמוד 69:

השורה:

```
char pattern[] = "ould";
```

מהווה הצהרה ואתחול של מערך. היא שקולה למעשה להצהרה:

```
char pattern[5];
```

שבעקבותיה סדרת ההוראות:

```
pattern[0] = 'o';
pattern[1] = 'u';
pattern[2] = 'l';
pattern[3] = 'd';
pattern[4] = '\0';
```

(יש לזכור כי מערכים ב-C מתחילים תמיד באינדקס 0.)

שורת האתחול הזו גורמת למהדר לספור את מספר התווים הדרוש (5 במקרה זה, כולל התו '\0' המציין סוף מחרוזת), להקצות זיכרון בגודל שאותו חישב, ולייצר את הקוד הדרוש לביצוע האתחול.

שתי השורות:

```
int getline (char line[ ], int max);
int strindex (char source[ ], char searchfor[ ]);
```

הן שורות אב-טיפוס (prototype). אנו **מצהירים** (declaration) באמצעותן כי:

א. הפונקציה getline מקבלת שני פרמטרים, מערך של תווים (char) ומספר שלם, ומחזירה מספר שלם.

ב. הפונקציה strindex מקבלת שני פרמטרים. כל אחד משני הפרמטרים הנו מערך של תווים. הפונקציה מחזירה מספר שלם.

האב-טיפוס של C מהווה אך ורק הצהרת כוונות; על המהדר לבדוק אם **טיפוסי** המשתנים (וטיפוסיהם בלבד) המועברים בעת הקריאה לפונקציה תואמים למעשה את טיפוסי המשתנים שניתנו בהצהרת האב-טיפוס, ואם השימוש בערך המוחזר נעשה בהתאם לטיפוס שהפונקציה אמורה להחזיר לפי ההצהרה. אם אחד משני התנאים אינו מתקיים, יוציא המהדר הודעת **אזהרה**. אולם אין זו שגיאה להעביר פרמטרים שלא בהתאם להצהרת האב-טיפוס. מטרת האב-טיפוס היא לעזור למתכנת לבדוק באופן אוטומטי את טיפוסי הנתונים המועברים, ולבצע בעקיפין cast על הפרמטר המועבר (כפי שהודגם בפרק 2, בדוגמה של sqrt(2)).

לשמות הפרמטרים שבהצהרת האב-טיפוס אין כל משמעות. כמו כן, אין הם חייבים להיות זהים לפרמטרים הפורמאליים המופיעים בהגדרת הפונקציה. למעשה, השמות הם אופציונליים. ניתן להשמיטם ולכתוב:

```
int getline(char[], int);
int strindex(char[], char[]);
```

מובן שעדיף להשאיר את השמות וכן להקפיד על שמות משמעותיים, לשם בהירות הקוד.

כאשר רוצים להצהיר על פונקציה שאינה מקבלת פרמטרים כלל, כותבים:

```
int func(void)
```

בדומה, פונקציה שאינה מחזירה ערך תיכתב:

```
void func(…)
```

סדר הפונקציות בקובץ אינו חשוב. אילו היינו כותבים את הפונקציה main בסוף הקובץ במקום בתחילתו, היה המהדר נתקל בהגדרות של הפונקציות getline

ו-strindex עוד לפני שהיה מגיע לפונקציה main. במקרה כזה אפשר היה להשמיט את שתי שורות האב-טיפוס, היות שטיפוסי הפרמטרים של הפונקציות getline ו-strindex היו כבר ידועים בזמן הידור ה-main, וכך יכול היה המהדר להתריע על אי-התאמה בטיפוסים, גם ללא כתיבה מפורשת של האבות-טיפוס.

יש לשים לב לשימוש באופרטור הפסיק בפונקציה strindex. האתחול של j ו-k, והקידום שלהם קשורים יחד מבחינה רעיונית. (כל אחד מהם מחזיק את האינדקס במערכים s ו-t בהתאמה, ואת התווים הנמצאים במערך באינדקסים j ו-k משווים זה לזה).

ב-C ניתן לכתוב את משפט ה-return בכל מקום שבו חוקי לכתוב statement.

בסעיף זה מוצג המבנה הכללי של תכנית בשפת C: תכנית היא אוסף של פונקציות הנמצאות על פני כמה קבצים מתוך אוסף הקבצים של מערכת ההפעלה. פונקציה אחת מיוחדת בכך ששמה הוא main, וממנה מתחיל ביצוע התכנית. למעט בחינה זו, main היא פונקציה רגילה, וככזאת ניתן אף לקרוא לה מפונקציה אחרת.

ניתן לקרוא את הערך ש-main מחזירה אל מערכת ההפעלה באמצעות משתנה מתאים של סביבת המערכת. התכנית יכולה להחזיר ערכים שונים, על פי נקודות יציאה שונות. הדבר יכול לשמש למשל לסימון שגיאות, שאותן רוצים להעביר למערכת ההפעלה. במערכות הפעלה שונות, משמעות משתנה זה יכולה להיות אחרת ותלויה בסביבת העבודה.

דוגמה

בדוגמה הבאה נרצה לסמן למערכת ההפעלה כי קריאה של תו מהקלט נכשלה.
במקרה של כישלון, מחזירה התכנית לסביבת מערכת ההפעלה את הערך 1.
במקרה של הצלחה, מחזירה התכנית לסביבת מערכת ההפעלה את הערך 0.

כדי להבין כיצד תשתמש סביבת העבודה בערכים אלו, יש לקרוא את יחידות הלימוד: "מדריך לסביבת UNIX" ופרק 8: "The UNIX System Interface" (שם נראה כי הסימון הוא בדרך כלל הפוך: ממערכת ההפעלה אל התכנית).

```
#include <stdio.h>
int main ( )
{
   int read_char;  /* char value from input */
   if ((read_char = getchar( )) == EOF)
      return 1; /* read fail */
      else
      {
            putchar(read_char);          /* normal case */
            return 0;
      }

}
```

הסמנטיקה של שפת C אינה דורשת שמספר הפרמטרים האקטואליים וטיפוסיהם בעת הקריאה לפונקציה יהיה זהה בהתאמה למספר הפרמטרים הפורמאליים ולטיפוסיהם. אם האב-טיפוס של הפונקציה ידוע למהדר, הוא יתריע במקרה של חוסר זהות, אך לא יפסול זאת.

כדי לקבל קוד טוב, אסור להתעלם מהודעות האזהרה של המהדר. ברוב המקרים מלמדת האזהרה על שגיאת תכנות. לעיתים מצביעה האזהרה על בעיה בניידות הקוד. לכן יש לשנות את התכנית, כך שלא נקבל אזהרות כלל.

כפי שתואר בספר, תכנית C יכולה להשתרע על פני כמה קבצים, וניתן לבצע הידור של כל הקבצים ביחד או לחוד.

השימוש במהדר למטרה זו מתואר במדריך נפרד, המלווה את סביבת הפיתוח.

כאשר מהדרים סדרת קבצים, יש לזכור נקודה חשובה: בזמן שהמהדר עובר מקובץ לקובץ, **אין** הוא שומר כל מידע הקשור בקובץ הקודם, כדי להשתמש בו עבור הקובץ הבא – זיכרונו נקי **לחלוטין**. אולם במסגרת **הפונקציה** העוברת הידור כעת, המהדר זוכר את הגדרות כל הפונקציות הקודמות לה, ולכן אין צורך להצהיר עבורן על

האבות-טיפוס בכל קריאה וקריאה. אם מעוניינים להצהיר על אבות-טיפוס לגבי פונקציות שבהן משתמשים בקובץ הנוכחי, אך אינן מוגדרות בו, ההצהרות חייבות להופיע בקובץ לפני הקריאה להן. הדבר נעשה בכוונה תחילה: הצהרת הפונקציות מראש מאפשרת אי-תלות במימושן.

יש להשתדל להבחין בין הצהרה על פונקציה ובין הגדרתה. הצהרה על פונקציה היא למעשה השורה הראשונה של הגדרתה: כאן מתואר אילו פרמטרים מקבלת הפונקציה, מה היא מחזירה ומה הם האבות-טיפוס של ערכים אלה. ההגדרה כוללת, נוסף על כך, גם את קטע הקוד המממש את פעולת הפונקציה.

אם לא מצהירים מה הפונקציה מחזירה, המהדר יניח **מראש** שהיא מחזירה int.

לא מומלץ לכתוב פונקציה אשר לפעמים מחזירה ערך ולפעמים לא. לדוגמה:

```
int f (int x)
{
      if (x > 0)
         return(17);
}
```

אם יש עניין בתוצאה שהפונקציה מחזירה, יש לדאוג שהיא תחזיר ערך בכל מקרה. אם אין מעוניינים בתוצאה, אלא רק בתוצאות הלוואי של הגדרתה, יש לכתוב return בלבד, ללא ביטוי.

שגיאת תכנות נפוצה היא להוסיף את התו ';' בשורה הראשונה של הגדרת הפונקציה. לדוגמה:

זו שגיאה!

```
int f (int x);
{
      if (x > 0)
         return(17);
      else
         return(–17);
}
```

השורה הראשונה בדוגמה זו צריכה להיות, כמובן: int f (int x) (ללא התו ';') שגיאת תכנות זו גורמת למהדר להניח שאנו פשוט מצהירים על הפונקציה, ובהמשך תתקבל מהמהדר הודעת שגיאה אשר תצביע על מקום שייראה (ובדרך כלל גם יהיה) חוקי לחלוטין.

מרבית האבות-טיפוס לפונקציות השימושיות נמצאים בקבצים stdio.h,stdlib.h ו-math.h. באמצעות Appendix-B ניתן לדעת באיזה קובץ header נמצא האב-טיפוס של כל פונקציה סטנדרטית.

הערה: כמו כן, ניתן להשתמש בתכנית שירות בשם grep שמטרתה היא לחפש מחרוזת בקובץ. למעשה, מחפשת grep ביטוי רגולרי .

תזכורת:

ביטוי רגולרי (A regular expression)

ביטוי רגולרי הוא ביטוי המתאר קבוצה של מחרוזות תווים (strings). השימוש בביטוי זה, נעשה כדי לתת תיאור עקבי של קבוצה, מבלי לפרט ברשימה את כל מרכיביה. לדוגמה, את הקבוצה המכילה את המחרוזות: baddef, bcddef **ניתן לתאר בביטוי:** "b(a|c)ddef" .

תרגיל 19

יש לבצע את תרגיל 4-1: נדרש לכתוב את הפונקציה strindex(s,t) המחזירה את מיקומו של המופע הימני ביותר של המחרוזת t במחרוזת s, או 1− אם אין מופע כזה.

פתרון

```
#include <stdio.h>
int strindex (char s[], char t[])
/* Find right most occurrence of string t in string s */
{
    int return_val = –1;
    char s_ndx, t_ndx, tmp_s;

    for (s_ndx = 0; s[s_ndx] != '\0'; s_ndx++)
```

```
    {
        for (t_ndx = 0, tmp_s = s_ndx; t[t_ndx] = = s[tmp_s]
                && t[t_ndx] != '\0'; t_ndx++, tmp_s++)
                ;
        if (t[t_ndx] = = '\0')
                return_val = s_ndx;
    }

    return (return_val);
}
main ( )
{
    char search_st[] = "hello bye bye hello";
    char search_for[] = "hello";

    printf ("right most occurance of");
    printf (" \"%s\" in \"%s\" is at index %d\n",
        search_for,
        search_st,
        strindex(search_st, search_for));
}
```

4.2 Functions Returning Non-integers

בפרק 2 דנו במה שעלול לקרות כאשר לא מספקים הצהרה נכונה לגבי הערך החוזר מפונקציה. הדגמנו זאת לגבי הפונקציה sqrt.

עתה נקדיש מספר מילים למילת המפתח void: למילת המפתח void שלושה שימושים עיקריים:

א. לציין פונקציה שאינה מחזירה כל ערך. לדוגמה:

```
void pr_X (int x)
{
        printf ("%d\n", x);
}
```

במקרה כזה, אם נכתוב בתוך הפונקציה return (expression); כאשר expression הוא ביטוי חוקי בשפה, תיווצר הודעת אזהרה של המהדר.

ב. לציין שהפונקציה היא חסרת ארגומנטים, כגון getchar או rand. ההגדרה נעשית בצורה:

```
int ret_7(void)
{
    return(7);
}
```

ג. לציין במפורש שאנו מתעלמים מהערך שהפונקציה מחזירה. זאת עושים על ידי cast על הערך החוזר ל-void. למשל, נניח שבשלב מסוים בתכנית אנו קוראים תו מתוך הקלט, אולם לא התו הנקרא עצמו מעניין אותנו, אלא רק עצם ההתקדמות בקלט. ניתן לכתוב זאת כך:

```
.
.
.
getchar( );
.
.
.
```

אולם בצורת הכתיבה הבאה:

```
.
.
.
(void) getchar( );
.
.
.
```

צורה זו עדיפה, מכיוון שכך אנו מיידעים את קורא התכנית, שאין כאן טעות של ממש (כפי שעלול להשתמע מן הצורה הראשונה), אלא בכוונה תחילה, אין אנו מעוניינים בערך החוזר. קורא התכנית יכול להיות אדם, אך גם תכנית אוטומטית כלשהי (lint -

פירוט נוסף לגבי תוכנה זו, בהמשך) אשר מבצעת אימות (verification) של תכנית ב-C. אם לא נבצע cast ל-void, תוציא תכנית כזאת הודעת אזהרה על כך שאנו מתעלמים מהערך ש-getchar מחזירה.

ל-void שימוש נוסף עבור מצביעים, אולם על כך נדון בפרק 5.

הסבר נוסף : תוכנת Lint

Lint הוא כלי תוכנה, אשר מטרתו להצביע על נקודות "חשודות" בקוד התכנית, כגון קבועים שאינם ניידים (portabe), פנייה למשתנים לפני אתחולם, תנאים המכילים **תמיד** ערכי true/false ועוד. בדרך כלל מתגלות הנקודות הנ"ל כטעויות תכנות.

<u>שימו לב:</u> קיימים שני ערכים מקובלים המסמנים טעות: 1– או 0 . יש להבחין ביניהם. למשל, יש לבדוק אם הפונקציה מחזירה char או unsigned char .

אם רוצים לבחון את אופן הביצוע של הפונקציה atof המוגדרת בעמוד 71, יש לקרוא לה בשם אחר (למשל atf). זאת מכיוון שהספרייה הסטנדרטית כבר מכילה פונקציה בשם זה, ולכן בזמן הקישור עלולה להיווצר התנגשות בין ה-atof החדש לבין זו של הספרייה (כך גם לגבי atoi).
נעיר כי הפונקציה isdigit בדוגמה זו שבספר, מקבלת כפרמטר תו אחד, ומחזירה אפס אם התו איננו אחת מהספרות 0,1,2, ...9.

4.3 External Variables

תרגיל 20

יש לכתוב ולהריץ את תכנית המחשבון – calculator המופיעה בספר (החל מעמוד 76). שימו לב כי הספרייה הסטנדרטית של C כבר מכילה פונקציה בשם ungetch, לכן יש להשתמש בשם אחר.

בסעיף זה, חשוב מאוד להבין את הנקודה הבאה:
כל משתנה המוגדר מחוץ לפונקציה הוא משתנה חיצוני (external). זאת אומרת, כל הפונקציות המופיעות החל **ממקום** ההגדרה בקובץ ועד לסוף הקובץ, יכולות לגשת אל משתנה זה, לקרוא אותו ולכתוב לתוכו. למעשה, כל דבר ב-C שעליו מצהירים מחוץ לפונקציה, הוא גלובלי, במובן זה שהוא מוכר מעתה ועד לסוף הקובץ (כמו למשל, אב-טיפוס). הואיל ואין אפשרות לכתוב פונקציה של פונקציה, נובע שכל הפונקציות הן

גלובליות. לכן כל פונקציה ב-C יכולה לקרוא לכל פונקציה אחרת. (זוהי למעשה ברירת המחדל. בהמשך נראה כיצד אפשר לשנות את תחום ההכרה – scope של פונקציה, ולהגביל את הפונקציות שיכולות לקרוא לה.)

הפורמט %g ב-printf מסייע אף הוא (כמו %f) להדפסת מספרים ממשיים. הסבר על כך נמצא ב-Appendix-B, בעמוד 244.

תרגיל 21

יש לבצע את תרגיל 3-4: בהינתן המסגרת הבסיסית, יש להרחיב את תכנית המחשבון calculator על ידי הוספת אופרטור מודולו (%) ותנאים עבור מספרים שליליים. יש לבצע cast ל-int עבור האופרנדים של %.

פתרון

(שימו לב לשינויים ביחס לתכנית המקורית המופיעה בספר.)

```
#include <stdio.h>
#include <math.h>        /* for atof */
#include <stdlib.h>

#define MAXOP 100        /* max size of operand or operator */
#define NUMBER '0'       /* signal that a number was found */

int getop (char []);
void push (double);
double pop (void);

/* reverse Polish calculator */
main ( )
{
    int type;
    double op2;
    char s[MAXOP];

    while ((type = getop(s)) != EOF)
```

```
switch (type)
{
        case NUMBER:
                push (atof(s));
                break;
        case '+':
                push (pop( ) + pop ( ));
                break;
        case '*':
                push (pop( ) * pop( ));
                break;

        case '–':
                op2 = pop( );
                push (pop( ) – op2);
                break;
        case '/':
                op2 = pop( );
                if (op2 != 0.0)
                        push (pop( ) / op2);
                else
                        printf("Error: zero devisor\n");
                break;
        case '%':
                op2 = pop( );
                if (op2 != 0.0)
                        push ((int) pop( ) % (int) op2);
                else
                        printf("Error: zero devisor\n");
                break;
        case '\n':
                printf ("\t%.8g\n", pop( ));
                break;
        default:
```

```
                printf ("Error: unknown command\n");
                break;
        }
    return 0;
}

#define MAXVAL 100          /* max depth of val stack */

int sp = 0;                 /* next free stack position */
double val [MAXVAL];        /* value stack */



/* push: push f onto value stack */
void push (double f)
{
    if (sp < MAXVAL)
        val[sp++] = f;
    else
        printf ("Error: stack is full, can't push %g\n", f);
}

/* pop: pop and return top value from stack */
double pop(void)
{
    if (sp > 0)
        return (val[– –sp]);
    else
    {
        printf ("Error: stack empty\n");
        return 0.0;
    }
}
```

```
#include <ctype.h>

int my_getch(void);
void my_ungetch (int);

/* getop: get next operator or numeric operand */
int getop (char s[])
{
    int i, c;

    while ((s[0] = c = my_getch( )) = = ' ' | | c = = '\t')
        ;
    s[1] = '\0';
    if (!isdigit(c) && c != '.' && c != '–')
        return c;  /* not a number */
    i = 0;
    if (isdigit(c))   /* collect integer part */
        while (isdigit(s[++i] = c = my_getch( )))
                ;
    if (c = = '–')
    {
        while (isdigit(s[++i] = c = my_getch( )))
                ;
        if (i = = 1 && c != '.')       /* minus sign alone */
        {
                my_ungetch(c);
                return '–';
        }
    }
    if (c = = '.')     /* collect fraction part */
        while (isdigit(s[++i] = c = my_getch( )))
                ;
    s[i] = '\0';
    if (c != EOF)
```

```
        my_ungetch (c);
    return NUMBER;
}

#define BUFSIZE 100

char buf[BUFSIZE];    /* buffer for ungetch */
int bufp = 0;                /* next free position in buf */

int my_getch(void)  /* get a (possibly pushed back) character */
{
    return (bufp > 0 ? buf[– –bufp] : getchar( ));
}

void my_ungetch (int c) /*  push character back on input */
{
    if (bufp >= BUFSIZE)
        printf ("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```

תרגיל 22

יש לבצע את תרגיל 4-4: יש להרחיב את תכנית המחשבון-calculator על ידי הוספת הפקודות: הדפסת האלמנט בראש (top) המחסנית (מבלי לבצע pop), שכפול (duplicate) של האלמנט, החלפה (swap) בין שני האלמנטים הראשונים. כמו כן יש להוסיף פקודה לניקוי (clear) המחסנית.

סימון הפקודות יהיה: 't' עבור top, 'd' עבור duplicate , 's' עבור swap ו- 'c' עבור clear.

פתרון

```
#include <stdio.h>
#include <math.h>                              /* for atof */
```

```
#include <stdlib.h>

#define MAXOP 100         /* max size of operand or operator */
#define NUMBER '0'        /* signal that a number was found */
#define COMMAND 'C'       /* signal that a command was found */

int getop (char []);
void push (double);
double pop (void);
void top_print(void);
void dup_stack(void);
void swap_stack(void);
void clear_stack(void);

/* reverse Polish calculator */
main ( )
{
    int type;
    double op2;
    char s[MAXOP];

    while ((type = getop(s)) != EOF)
        switch (type)
        {
                case COMMAND:
                        if (s[0] = = 't')
                                top_print( );
                        else if (s[0] = = 'd')
                                dup_stack( );
                        else if (s[0] = = 's')
                                swap_stack( );
                        else if (s[0] = = 'c')
                                clear_stack( );
                        else
```

```
                printf("Unknown command %s\n", s);
        break;
case NUMBER:
        push (atof(s));
        break;
case '+':
        push (pop( ) + pop ( ));
        break;
case '*':
        push (pop( ) * pop( ));
        break;
case '–':
        op2 = pop( );
        push (pop( ) – op2);
        break;
case '/':
        op2 = pop( );
        if (op2 != 0.0)
                push (pop( ) / op2);
        else
                printf("Error: zero devisor\n");
        break;
case '%':
        op2 = pop( );
        if (op2 != 0.0)
                push ((int) pop( ) % (int) op2);
        else
                printf("Error: zero devisor\n");
        break;
case '\n':
        printf ("\t%.8g\n", pop( ));
        break;
default:
        printf ("Error: unknown command\n");
```

```
                    break;
        }
    return 0;
}

#define MAXVAL 100          /* max depth of val stack */

int sp = 0;                 /* next free stack position */
double val [MAXVAL];        /* value stack */

/* push: push f onto value stack */
void push (couble f)
{
    if (sp < MAXVAL)
        val[sp++] = f;
    else
        printf ("Error: stack is full, can't push %g\n", f);
}

/* pop: pop and return top value from stack */
double pop(void)
{
    if (sp > 0)
        return (val[– –sp]);
    else
    {
        printf ("Error: stack is empty\n");
        return 0.0;
    }
}


#include <ctype.h>
```

```
int my_getch(void);
void my_ungetch (int);

/* getop: get next operator or numeric operand */
int getop (char s[])
{
    int i, c;

    while ((s[0] = c = my_getch( )) = = ' ' | | c = = '\t')
        ;
    s[1] = '\0';
    if (!isdigit(c) && c != '.' && c != '–' && !(isalpha(c)))
        return c;  /* not a number */
    i = 0;
    if (isdigit(c))    /* collect integer part */
        while (isdigit(s[++i] = c = my_getch( )))
                ;
        s[i] = '\0';
        if (c != EOF)
                my_ungetch (c);
        return COMMAND;
    }


    if (isdigit (c))
        while (isdigit(s[++i] = c = my_getch( )))
                ;
    if (c = = '–')
    {
        while (isdigit(s[++i] = c = my_getch( )))
                ;
        if (i = = 1 && c != '.')   /* minus sign alone */
        {
                my_ungetch(c);
```

```
            return '–';
        }
    }
    if (c = = '.')     /* collect fraction part */
        while (isdigit(s[++i] = c = my_getch( )))
            ;
    s[i] = '\0';
    if (c != EOF)
        my_ungetch (c);
    return NUMBER;
}

#define BUFSIZE 100

char buf[BUFSIZE];    /* buffer for ungetch */
int bufp = 0;         /* next free position in buf */

int my_getch(void)    /* get a (possibly pushed back) character */
{
    return (bufp > 0 ? buf[– –bufp] : getchar( ));
}

void my_ungetch (int c)  /*  push character back on input */
{
    if (bufp >= BUFSIZE)
        printf ("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

void top_print(void)  /*  print the top element of the stack */
{
    if (sp > 0)
        printf ("%g\n", val[s–1]);
```

```
    else
        printf ("top: empty stack\n");
}

void dup_stack(void)  /*  duplicate the top element of the stack */
{
    if (sp > 0)
        push (val[sp – 1]);
    else
        printf ("dup: empty stack\n");
}

void swap_stack(void)  /*  swap the top two elements of the stack */
{
    if (sp > 1)
    {
        double tmp;
        tmp = val[so – 2];
        val[sp – 2] = val[sp – 1];
        val[sp – 1] = tmp;
    }
    else
        printf ("swap: less then 2 values in stack\n");
}

void clear_stack(void)  /*  clear the stack */
{
    sp = 0;
}
```

4.4 Scope Rules

כל משתנה ב-C יכול להיות מוגדר בתוך פונקציה או מחוץ לפונקציה. משתנה המוגדר בתוך פונקציה נקרא **משתנה אוטומטי** (גם הפרמטרים האקטואליים של פונקציה הם אוטומטיים). הקצאת הזיכרון למשתנה זה נעשית בד בבד עם הקריאה לפונקציה, והזיכרון הדרוש מוקצה על מחסנית זמן הריצה. ניתן להתייחס למשתנה כזה רק תוך כדי ביצוע הפונקציה.

משתנה המוגדר מחוץ לפונקציה נקרא **משתנה חיצוני** (גלובלי). הקצאת זיכרון למשתנה חיצוני נעשית כבר בזמן ההידור עצמו בחלק הנתונים -data- של התכנית. משך חייו הוא משך חייה של התכנית עצמה, וניתן להתייחס אליו בכל שלב של ריצת התכנית.

משתנה אוטומטי (מקומי), הזהה בשמו למשתנה חיצוני, "מסתיר" את המשתנה החיצוני. כלומר, כל אזכור של המשתנה בתוך הפונקציה ייוחס למשתנה האוטומטי, ולא למשתנה החיצוני. מובן שרצוי להימנע משימוש בשם זהה, לשם בהירות הקוד וכדי למנוע בלבול.

ניתן לגשת למשתנה חיצוני (גלובלי) שהוגדר בקובץ מסוים גם מפונקציות אחרות הנמצאות על קבצים אחרים. לצורך כך יש להשתמש במילת המפתח external. המילה external מודיעה למהדר שהמשתנה, ששמו וטיפוסו כתובים מיד אחריה, **מוגדר** כבר בקובץ אחר, ואין צורך להקצות עבורו שטח בזיכרון. כל התייחסות אל משתנה שהוצהר כ-external באמצעות שמו תתפרש כהתייחסות אל אותו משתנה שהוגדר.

על משתנה חיצוני מותר להצהיר ללא הגבלה: ניתן להצהיר עליו בכל קובץ שיש בו פונקציות המעוניינות לגשת למשתנה זה. אולם חייבת להיות **הגדרה אחת** ויחידה של המשתנה, באחד מקובצי התכנית. (הדרישה להגדרה אחת בלבד היא חידוש של תקן ANSI. בגרסה הראשונה של השפה הדבר לא היה הכרחי והתוצאות היו תלויות בתכנית הקישור (linker)).

מומלץ להימנע, ככל האפשר, משימוש במשתנים גלובליים, ובמקומם להשתמש בהעברת פרמטרים בין פונקציות. באופן זה תוכל התכנית להסתיר נתונים ומידע פנימי מפני פונקציות אשר אין להן צורך במידע זה.

תרגיל 23

יש לפצל את תכנית ה-calculator לשני קבצים:
בקובץ אחד תהיה הפונקציה main' ובשני – הפונקציות push, pop ו-getop.
הקובץ המכיל את main יכיל את ההגדרה של val, והקובץ השני יכיל את ההגדרה של sp.

פתרון

```
/* This is the main part of calc.c */

#include <stdio.h>
#include <math.h>     /* for atof */
#include <stdlib.h>

#define MAXOP 100       /* max size of operand or operator */
#define NUMBER '0'      /* signal that a number was found */
#define MAXVAL 100      /* max depth of val stack */

int getop (char []);
void push (double);
double pop (void);

double val[MAXVAL];

/* reverse Polish calculator */
main ( )
{
    int type;
    double op2;
    char s[MAXOP];

    while ((type = getop(s)) != EOF)
        switch (type)
        {
```

```
            case NUMBER:
                    push (atof(s));
                    break;
            case '+':
                    push (pop( ) + pop ( ));
                    break;
            case '*':
                    push (pop( ) * pop( ));
                    break;
            case '–':
                    op2 = pop( );
                    push (pop( ) – op2);
                    break;
            case '/':
                    op2 = pop( );
                    if (op2 != 0.0)
                            push (pop( ) / op2);
                    else
                            printf("Error: zero devisor\n");
                    break;
            case '\n':
                    printf ("\t%.8g\n", pop( ));
                    break;
            default:
                    printf ("Error: unknown command\n");
                    break;
        }
    return 0;
}

/* This is the second part. It contains the stack manipulation programs */

#include <stdio.h>
```

```
#define NUMBER '0'      /* signal that a number was found */
#define MAXVAL 100      /* max depth of val stack */

static int sp = 0;
extern val[];

/* push: push f onto value stack */
void push (double f)
{
    if (sp < MAXVAL)
        val[sp++] = f;
    else
        printf ("Error: stack is full, can't push %g\n", f);
}
/* pop: pop and return top value from stack */
double pop(void)
{
    if (sp > 0)
        return (val[– –sp]);
    else
    {
        printf ("Error: stack is empty\n");
        return 0.0;
    }
}

#include <ctype.h>

int my_getch(void);
void my_ungetch (int);

/* getop: get next operator or numeric operand */
int getop (char s[])
{
```

```
    int i, c;

    while ((s[0] = c = my_getch( )) = = ' ' | | c = = '\t')
        ;
    s[1] = '\0';
    if (!isdigit(c) && c != '.' && c != '–')
        return c;  /* not a number */
    i = 0;
    if (isdigit(c))   /* collect integer part */
        while (isdigit(s[++i] = c = my_getch( )))
                ;
    if (c = = '.')      /* collect fraction part */
        while (isdigit(s[++i] = c = my_getch( )))
                ;
    s[i] = '\0';
    if (c != EOF)
        my_ungetch (c);
    return NUMBER;
}

#define BUFSIZE 100

static char buf[BUFSIZE];  /* buffer for ungetch */
static int bufp = 0;               /* next free position in buf */

int my_getch(void)  /* get a (possibly pushed back) character */
{
    return (bufp > 0 ? buf[– –bufp] : getchar( ));
}

void my_ungetch (int c)  /*  push character back on input */
{
    if (bufp >= BUFSIZE)
        printf ("ungetch: too many characters\n");
```

```
    else
        buf[bufp++] = c;
}
```

4.5 Header Files

קובצי ה-header הסטנדרטיים (המפורטים ב-Appendix-B) מכילים קבועים סימבוליים, שהוגדרו על ידי #define, ואבות-טיפוס של פונקציות. ההוראה #include גורמת למהדר לקרוא את הקובץ המופיע בהמשך ההוראה ולהדר אותו. (למעשה, התהליך הוא קצת יותר מסובך, כפי שנראה בסעיף 4.11). תוכן קובץ ה-header יכול להכיל גם הגדרות של משתנים ושל פונקציות, אולם הדבר אינו מומלץ משתי סיבות:

א. לא מקובל לכתוב את גוף הפונקציות בקבצים בעלי סיומת h. כתיבה אחרת עלולה לגרום לחוסר בהירות.

ב. לא כל תכניות הנפוי (debuggers) מסוגלות להתמודד עם קוד מקור, הנמצא בקובץ h.

לכן נתבקש לתת בקובץ header רק הצהרות על פונקציות והגדרות של קבועים סימבוליים.

אם נרצה לדעת באיזה קובץ header מצוי האב-טיפוס של פונקציית הספרייה, שבו אנו רוצים להשתמש, מומלץ להשתמש במדריך של סביבת העבודה (במקרה שלנו יש לכתוב: man <שם פונקציה>). נקבל הסבר קצר על הפונקציה ונדע באיזה קובץ header מופיע האב-טיפוס שלה. לעיתים מוצהר האב-טיפוס ביותר מקובץ header אחד. במקרה זה יש לבחור תמיד בקובץ header סטנדרטי, אחד מאלו המופיעים בעמוד 241 בספר.

4.6 Static Variables

כפי שראינו בסעיף 4.4, יש משמעות למקום בקובץ שבו הגדרנו או הצהרנו על משתנה. המקום קובע באיזה חלק של הזיכרון יוקצה המשתנה: אזור הנתונים – data – או אזור המחסנית - stack, ולמי תהיה גישה אליו.

ההצהרה על משתנה כ-static מוסיפה גמישות ושליטה על תחום ההכרה של משתנה: כאשר מצהירים על משתנה אוטומטי של פונקציה כ-static, ניתן אמנם לגשת אליו מתוך הפונקציה בלבד, אולם הזיכרון שיוקצה עבורו יהיה בחלק ה-data של התכנית, ולכן ערכו לא ייהרס כאשר הפונקציה תסתיים. (שימו לב: הדבר משפיע על דרישות הזיכרון של התכנית.)

כאשר מצהירים על משתנה חיצוני כ-static, אזי הוא מוכר לכל הפונקציות ממקום ההצהרה ועד לסוף הקובץ, אולם לא ניתן להתייחס אליו מקובץ אחר (על ידי הצהרת external). זאת היות שהמהדר לא משאיר כל מידע עליו עבור תכנית הקישור.

הצהרה על פונקציה כ-static גורמת לתופעה דומה: כרגיל פונקציה היא ישות חיצונית באופיה – כלומר ניתן לקרוא לה מכל קובץ בתכנית. אולם כאשר נצהיר על פונקציה כ-static , רק פונקציות המוגדרות בקובץ יוכלו לקרוא לה.

תרגיל 24

יש לבצע את תרגיל 4-11: יש להוסיף פקודות לתוכנת המחשבון, כך שנקבל טיפול במשתנים. (קל להוסיף 26 משתנים כמספר אותיות ה-abc, כך נקבל שמות בעלי אות אחת.) יש להוסיף משתנה עבור הערך שהודפס אחרון.
זהו "getop" המקורי. יש להניח שלא תיתכנה שתי קריאות עוקבות ל-ungetch ללא getch ביניהן.

פתרון

```
/* getop: get next operator or numeric operand */
int getop (char s[])
{
   int i, c;
   static int last_read = 0;  /* the most recently printed value */

   if (last_read = = 0)
      c = my_getch( );
   else
   {
      c = last_read;
```

```
        last_read = 0;
    }

    while ((s[0] = c = my_getch( )) = = ' ' | | c = = '\t')
        c = my_getch( );
    s[1] = '\0';
    if (!isdigit(c) && c != '.')
        return c;  /* not a number */
    i = 0;
    if (isdigit(c))    /* collect integer part */
        while (isdigit(s[++i] = c = my_getch( )))
                ;
    if (c = = '.')      /* collect fraction part */
        while (isdigit(s[++i] = c = my_getch( )))
                ;
        s[i] = '\0';
    if (c != EOF)
        last_read = c;
    return NUMBER;
}
```

Recursion 4.10

תרגיל 25

יש לבצע את תרגיל 4-13: לכתוב גרסה רקורסיבית לפונקציה reverse(s), אשר מבצעת היפוך (reverse) של המחרוזת התווית s.

פתרון

```
#include <stdio.h>

/* reverse string s from start to end */
void reverse1(char s[], int start, int end)
```

```
{
    if (end - start >= 1)  /* if not empty */
    {
        char tmp;
        tmp = s[start];
        s[start] = s[end];
        s[end] = tmp;
        reverse1(s, start + 1, end – 1);
    }
}

/* reverse string s in place */
void reverse(char s[])
{
    int st_len;
    /* find string length */

    for (st_len = 0; s[st_len] != '\0'; st_len++)
        ;
    reverse1 (s, 0, st_len – 1);
}
/* reverse two strings */
main ( )
{
    char s[] = "Hello there";
    char t[] = "How are you?";

    printf("before reverse s=%s. ", s);
    reverse (s);
    printf("after reverse s=%s. ", s);
    reverse (t);
    printf("after reverse t=%s\n", t);
}
```

The C Preprocessor 4.11

תהליך ההידור ב-C כולל שלושה שלבים. הראשון הוא ה-cpp (C Pre-Processor) הגורם לקריאה של ה-include files ולהרחבה (expansion) של קטעי מקרו. הפלט של שלב זה מועבר לשלב השני – למהדר, המתרגם את התכנית, ראשית לשפת אסמבלי ואחר-כך לקובץ object. השלב האחרון הוא קישור של כל קובצי ה-object והספריות על ידי תכנית הקישור (linker).
למשל, כאשר עובדים בסביבת UNIX עם gcc, ללא דגלים כלשהם, מופעלים כל שלושת השלבים ברצף. gcc מכיל בתוכו cpp, מהדר ותכנית קישור.

ניתן לבצע כל אחד משלושת השלבים באופן נפרד. למשל, את ה-cpp של UNIX, ניתן להפעיל רק מתוך ה-commandline של ה-shell. זוהי תכנית שימושית מאוד. לעיתים מסתירים קטעי המקרו את הצורה האמיתית שבה רואה המהדר את הקוד. על ידי שימוש ב- cpp ניתן לקבל את המקור המגיע למהדר עצמו, ולבדוק אם טמונה בו טעות. כמו כן ניתן להשתמש באופציה של gcc כדי לקבל את אותה תוצאה .

דוגמה 1

cpp – קריאה של ה-include files:

נתונה תכנית המכילה את הקבצים:

קובץ name.c:

```
#include "a.h"
#include “b.h”
void main(void)
{
.....
}
```

קובץ a.h:

```
typedef char * string
#define shoe 1
#define table 2
```

קובץ b.h:

```
#define chair 3
#define map 4
```

הקובץ שנקבל לאחר פעולת ה-cpp יהיה שקול לקובץ המכיל:

```
typedef char * string
#define shoe 1
#define table 2
#define chair 3
#define map 4
void main(void)
{
.............
}
```

דוגמה 2

cpp – הרחבה (expansion) של קטעי מקרו:

נתון קובץ תכנית המכיל:

```
#define table 0
#define chair 1
#define computer 2

#define max (x, y) (x) > (y) ? (x) : (y)
.......
void main(void)
{
......
printf("%d",computer);
.....
if ( i>chair || i < table)
.........
```

```
printf("%d", max(a,b+2));
.....
}
```

הקובץ שנקבל לאחר פעולת ה-cpp יהיה שקול לקובץ המכיל:

```
void main(void)
{
......
printf("%d",2);
.....
if ( i>1 || i < 0)
.........
printf("%d", (a) > (b+2) ? (a) : (b+2));
.......
}
```

תרגיל 26

יש לבצע את תרגיל 14-4: להגדיר מקרו swap(t,x,y) אשר מחליף בין שני ארגומנטים מסוג t.

פתרון

```
/* interchange two arguments of type t */
#define swap(t, x, y)\
{  \
    t temp; \
    temp = x; \
    x = y; \
    y = temp; \
}
```

נקודות לסיכום פרק 4

הנושאים העיקריים שנדונו בפרק זה הם:

- פונקציות: אב-טיפוס (prototype), פרמטרים, ערך חזרה, הצהרה, הגדרה.
- תכנית המשתרעת על פני כמה קבצים.
- משתנה חיצוני (גלובלי) / משתנה אוטומטי (מקומי).
- קובצי header.
- משתנים קבועים / פונקציות קבועות (static).
- דוגמה לרקורסיה.
- שלבי ההידור ב-C.

תזכורת בפרק:

- ביטוי רגולרי (A regular expression)

מילון מונחים לפרק 4

prototype	אב-טיפוס
stack	אזור המחסנית (בזיכרון)
data	אזור הנתונים (בזיכרון)
declaration	הצהרה
automatic (local) variable	משתנה אוטומטי (מקומי)
external variable	משתנה חיצוני
scope	תחום ההכרה (של פונקציה)

פרק 5 Pointers and Arrays

5.1 Pointers and Addresses

בשפת C, האופרטור & המופיע לפני משתנה מחזיר ערך המייצג את **כתובתו** (addresses) של אותו משתנה בזיכרון. ואולם הדרך שבה מיוצגות הכתובות היא ייחודית לכל מהדר ולכל מכונה. זהו מקור שכיח לבעיות בניידות הקוד. לפיכך יש להשתמש במצביעים בזהירות, ולצמצם את השימוש בחישובי כתובות.

יש לשים לב: האופרטור & אינו מבצע הקצאה ככל האפשר, הוא נותן אך ורק את **הכתובת** של המשתנה. רק פונקציית הספרייה הסטנדרטית malloc() היא זו המבצעת הקצאה דינמית של זיכרון.

בשימוש באופרטור * יש להיות בטוחים שהערך המוכל במשתנה המצביע כבר מכיל **כתובת**, שמותר לכתוב בה. ב-C יש להיות זהירים, מכיוון שיש אפשרות לפנות **לכתובות** בצורה ישירה, באמצעות האופרטור &. כלומר, אפשר לפנות לכתובת, גם אם לא הוקצתה, וזהו, כמובן, מקור שכיח לטעויות תכנות.

נראה זאת על ידי הדוגמה שבתכנית הבאה:

```
/* chapter 5, first example.
Incorrect allocation of memory */

#include <stdio.h>

int *pointer_to_int;

func_a (void)
{
   int a = 77;
```

```
    pointer_to_int = &a;
}

func_b (void)
{

    long b;
    b = 555555;
}

main(void)
{

    func_a( );
    printf ("%d\n", *pointer_to_int);
    func_b( );
    printf ("%d\n", *pointer_to_int);
}
```

תרגיל

יש להריץ את התכנית, ולראות את תוצאות ההדפסה.

לאחר ההרצה, נראה כי שתי ההדפסות המופיעות בתכנית הראשית מדפיסות ערכים שונים, כלומר הקריאה לפונקציות שינתה את תוצאות ההדפסה.

תזכורת:

רשומת הפעלה (activation record)

רשומת הפעלה היא קטע זיכרון, המכיל את הארגומנטים ואת המשתנים המקומיים של פונקציה. משתני רגיסטר או משתנים סטטיים אינם מופיעים בזיכרון זה. כמו כן מכילה הרשומה את כתובת הזיכרון, שאליו תחזור התכנית לאחר סיום הפונקציה.

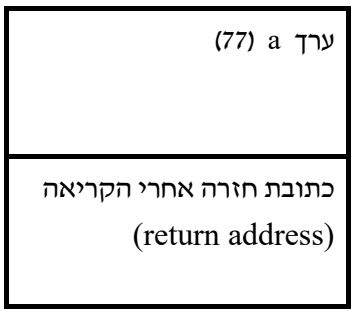
רשומת ההפעלה נוצרת באופן אוטומטי, כאשר מתבצעת קריאה לפונקציה, ומשוחררת, אוטומטית, כאשר הפונקציה מסתיימת. רשומות ההפעלה של כל הפונקציות הפעילות, מאוחסנות באזור הזיכרון הנקרא "מחסנית". ברגע ששוחררה רשימת הפעלה של פונקציה מסוימת, מתפנה מקום במחסנית לרשומות הפעלה חדשות, עבור פונקציות נקראות אחרות.

נסביר מה קרה בתכנית האחרונה: הפונקציה func_b לא משנה את הערך שיש למשתנה (הגלובלי) pointer_to_int. המשתנה ממשיך להחזיק בערך שקיבל לאחר הקריאה ל-func_a (כאן: כתובת המשתנה a). אם כך, מדוע יש שוני בין שתי ההדפסות? הסיבה היא ש**תוכן הזיכרון** שעליו מצביע pointer_to_int השתנה.
כיצד זה קרה? הסיבה לכך היא שהמשתנה a הוא משתנה אוטומטי, והוא הוקצה על זיכרון המחסנית בזמן הריצה. לכן הערך שקיבל pointer_to_int מצביע למעשה על **כתובת** במחסנית (שהיא כתובתו הזמנית של המשתנה a). לאחר סיום ביצוע הפונקציה func_a, המחסנית מתנקה, אולם pointer_to_int ממשיך להצביע על אותה כתובת, למרות שהוא כבר לא קשור אליה (מכיוון שהמשתנה a שוחרר, ולכן לא קיים יותר). בעת הקריאה ל-func_b נוצרת רשומת הפעלה חדשה במחסנית (תזכורת לגבי רשומת הפעלה, למעלה). השדות של הרשומה נכתבים גם הם בזיכרון, שעליו מצביע pointer_to_int (שכאמור מצביע על זיכרון במחסנית). כלומר, בזמן ההדפסה השנייה, רשומת ההפעלה מכילה את המשתנה המקומי b, ומכאן השוני בהדפסות. למעשה, גם ההדפסה הראשונה אינה נכונה **לוגית**, מאותה סיבה: מותר לבצע התייחסות ל-pointer_to_int רק בתוך func_a כולל בפונקציות ש-func_a קראה להן, והעבירה את ערכו של pointer_to_int כפרמטר. כלומר, כל עוד רשומת ההפעלה של func_a עדיין קיימת.
אילו a היה מוצהר כ-static, ההדפסות היו יוצאות זהות, מכיוון שאז היה a מוקצה על חלק ה-data של התכנית.

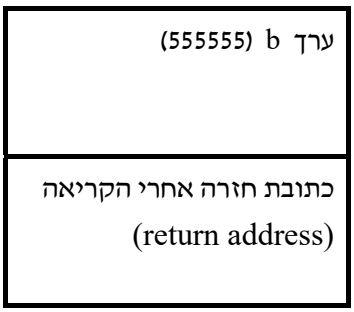
בדוגמה שלמעלה, כאשר תיקרא הפונקציה func_a, רשומת ההפעלה תכיל את המשתנה a , וכן את כתובת החזרה מהפונקציה. כאשר מסתיימת הפונקציה, מתנקה המחסנית, ואז מתבצעת הקריאה לפונקציה func_b. אז תכיל רשומת ההפעלה את המשתנה b ואת כתובת החזרה מהפונקציה. (בדוגמה זו אין קריאה של פונקציה מתוך פונקציה, לכן המחסנית מכילה רשומת הפעלה אחת בכל פעם.)

נראה זאת בסרטוטים הבאים:

מצב המחסנית לאחר הקריאה לפונקציה func_a: (המחסנית מתחילה בכתובת s1)



מצב המחסנית לאחר הקריאה לפונקציה func_b: (המחסנית מתחילה בכתובת s1)



בכל מצב מכיל המשתנה הגלובלי pointer_to_int את הכתובת s1:
לאחר הקריאה לפונקציה func_a, הכתובת s1 מכילה 77.
לאחר הקריאה לפונקציה func_b, הכתובת s1 מכילה 555555.
מכאן השוני בהדפסות הערך המופיע בכתובת שבמשתנה pointer_to_int:
ראשית מודפס 77, ואחר-כך 555555.

5.2 Pointers and Function Arguments

יש לזכור: **בשפת C, העברת הפרמטרים לפונקציות נעשית אך ורק לפי ערך.** כלומר, משתנים חדשים מוקצים על המחסנית, וערכי הפרמטרים **מועתקים** אליהם. המושג "פרמטר משתנה" (העברה לפי כתובת), לא קיים. אם נרצה שפונקציה תשנה את ערכו של פרמטר אקטואלי, יש להעביר את **כתובתו** של הפרמטר האקטואלי. ערך זה

(הכתובת) **יועתק** אל משתנה מקומי אשר יהיה מטיפוס מצביע (כמו px ו-py בדוגמת ה-swap בעמוד 96 בספר. ניתן להבין זאת גם באמצעות האיור המצורף לדוגמה).

תרגיל

יש לבצע את תרגיל 5-2: נדרש לכתוב את הפונקציה get_float, שהיא המקבילה לפונקציה getint (עמוד 97 בספר) עבור floating-point. כלומר, הפונקציה צריכה לקבל את ערך ה-floating-point הבא מהקלט, ולהכניסו למצביע fp*.
יש לשים לב לערך החזרה של הפונקציה.

יש להניח כי מספר ממשי חוקי מיוצג כרצף ספרות שאחריהן התו '.' ואחריו שוב רצף של ספרות. מובן שמספר ממשי יכול להיות גם ללא חלק שבור, או ללא חלק שלם (לדוגמה 0.5, .5, 5 הם מספרים ממשיים חוקיים).

פתרון (5-2)

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

#define BUFSIZE 100

static char buf[BUFSIZE];  /* buffer for ungetch */
static int bufp = 0;              /* next free position in buf */

int my_getch(void)  /* get a (possibly pushed back) character */
{
   return (bufp > 0 ? buf[– –bufp] : getchar( ));
}

void my_ungetch (int c)  /*  push character back on input */
{
   if (bufp >= BUFSIZE)
      printf ("ungetch: too many characters\n");
   else
      buf[bufp++] = c;
```

```
}

/* get next float from input into *fp */
float get_float (float *fp)
{
    int c, sign;
    long exp = 10;

/* There are 4 stages:
    a. skip white spaces
    b. decide whether there is a sign
    c. calculate part before the decimal point
    d. calculate part after the decimal point
*/

    while (isspace (c = my_getch( )))
    /* skip white spaces */
        ;

    if (!isdigit(c) && c != EOF && c != '+' &&
        c != '–' && c != '.')
    {
        my_ungetch(c);
              /* Not a valid beginning of floating point number */
        return 0.0;
    }

    sign = (c = = '–' ) ? –1 : 1;
    if (c = = '–' | | c = = '+')
        c = my_getch( );

    for (*fp = 0.0; isdigit(c); c = my_getch( ))
        *fp = 10 * *fp + (c – '0');
```

```
    if (c = = '.')
        for (c = my_getch( ); isdigit(c); c = my_getch( ), exp*=10)
                *fp = *fp + (float) (c – '0') / exp;

    *fp *= sign;

    if (c != EOF)
        my_ungetch (c);
    return c;
}

main( )
{
    float ret_val;

    get_float (&ret_val);
    printf("%g\n", ret_val);
}
```

5.3 Pointers and Arrays

שפת C מבטיחה שכל מערך יוקצה בגוש רציף בזיכרון. יש לזכור נקודה זו בסעיף הבא העוסק באריתמטיקה של מצביעים.

חשוב לזכור שחישוב כתובות של מערך נעשה תמיד על ידי אופרטור חיבור. המהדר מייצר את הקוד המבצע חיבור. וכך יוצא ש- a[i] זהה ל- *(a+i), ו-&a[i] זהה ל-a+i. המהדר מתייחס לסוגריים המרובעים ([]) כאל אופרטור חיבור. כאשר הטיפוס בתוך הסוגריים אינו שלם, מהדרים אחדים יתנו הודעה בנוסח

operands of + have incompatible types

כלומר, מתבצע חיבור של ממש.
היות שחיבור הוא פעולה חילופית (קומוטטיבית), כתיבת i[a] היא חוקית, ושקולה ל-a[i], אך לא מקובל לכתוב כך.

מותר (ולפעמים יש צורך) לתת אינדקסים שליליים. למשל, קטע הקוד הבא ידפיס את התו 'e':

```
/* chapter 5, second example
Usage of negative index */

#include <stdio.h>
main()
{
    char b[] = "hello";  /* automatic memory allocation */
    char *x;
    x = &b[4];
        /* Or one can write x = b + 4 which is the same */
    printf ("%c\n", x[–3]);
}
```

ב-C לא קיימת אפשרות של range checking – בדיקה של חריגה ממערך, כפי שמאפשרות שפות אחרות.

מערך ב-C הוא, למעשה, גוש רציף של זיכרון שכתובת תחילתו ידועה לנו. ב-C, כפי שמראה הדוגמה לעיל, גם x[0] וגם b[4] מתייחסים אל אותו איבר. המהדר לא מסוגל לעקוב אחר כל הצורות האפשריות, שבהן יכולים להתייחס אל איברי b (כגון x[0]), כדי לייצר קוד בדיקה, שהרי מעקב כזה כבר מצריך את הרצת התכנית.

חריגה ממערך עשויה לגרור אחריה תוצאות "מוזרות". התכנית תתייחס בזמן הריצה אל תוכן זיכרון שלא שייך לה, או כזה המכיל ערכים של משתנים אחרים. היא תנסה לפרש את התוכן של אותם משתנים, על פי סוג המשתנה שאליו היא מתייחסת. לדוגמה, נריץ את התכנית הבאה:

```
/* chapter 5, third example
Example - array overflow */

#include <stdio.h>
main()
{
```

```
    int x = 4444;
    char a[] = "hello";
        /* The size of a is six bytes, from a[0] to a[5] */
    char b[] = "there";
        /* This compiler will allocate space for "there" right after "hello" */

    printf ("%c\n", a[6]);
        /* Overflow. The maximum is a[5].
           We are getting the letter 't' of "there" */

    printf ("%c\n", a[−1]);
       /* Underflow. The minimum is a[0].
          We are getting some garbage due to the value of x */
}
```

אילו היינו נותנים ל-a[−1] מן הדוגמה הנ״ל ערך כלשהו (משפט השמה), היה ערכו של המשתנה x מושפע מכך (מכיוון שהוא מופיע במקום אחד לפני המערך a). גלישה ממערך מתגלה, בדרך כלל, בעת שמשתנים מסוימים מקבלים ערכים ״מוזרים״. יתר על כן, בעבודה עם המנפה (debugger) אפשר להיווכח, באמצעות מעקב (trace), שמשתנה אכן עשוי לשנות את ערכו, בעקבות ביצוע השמה למשתנה אחר. הדבר רומז, ברוב המקרים, על גלישה.

יש לשים לב: תוצאות ההרצה אינן בהכרח כפי שנכתב כאן, מכיוון שהדבר תלוי בארכיטקטורה של המחסנית, ומשתנה ממכונה למכונה. מומלץ להריץ את התוכנית ולבדוק בעצמכם!

<u>תרגיל:</u> נוסיף לדוגמה לעיל משפט המציב ב-a[−1] את הערך 'k' ונריץ את התכנית. כיצד יושפע המשתנה x? האם ערכו יישאר 4444?

חשוב לשים לב להבדל שבין a ל-pa, כפי שהם מוגדרים בספר בעמוד 98:

```
int  a[10];
int  *pa;
```

a הוא **קבוע**, והוא משמש למעשה כתווית (label) של המקום בזיכרון. בכל מקום שבו נכתוב a, נכתב הערך של התווית a, כלומר המקום בזיכרון שהחל ממנו הוקצה מקום בגודל המתאים. pa, לעומת זאת, הוא **משתנה**. (לכן השמה ל-a אינה אפשרית בעוד

שהשמה ל-pa חוקית). מוקצה לו מקום בזיכרון, ובו נשמר הערך שאותו יש למצביע. כאשר כותבים pa[i], המהדר מייצר קוד, אשר דואג לפנות אל המשתנה הנמצא i ״יחידות״ מן המשתנה שעליו מצביע pa.

אולם בעת קריאה לפונקציה, אין כל הבדל אם הפרמטר הפורמלי הוצהר כ-sometype *p או כ-sometype[]. שפת C מקצה, תמיד, מקום לפרמטר על המחסנית, ומעתיקה אליו את הערך של המשתנה האקטואלי (כלומר, המשתנה של הפונקציה הנקראת עכשיו). לכן אין הבדל בין קריאה של pa כפרמטר אקטואלי, או בקריאה של a כפרמטר אקטואלי. המהדר ייצר קוד, שיעתיק את הערך הנקרא אל המשתנה האוטומטי בזיכרון. היות שזהו משתנה, ניתן לשנות את ערכו, אפילו אם הפרמטר האקטואלי שהועבר הוא a (קבוע). יש לזכור: אין משנים את הקבוע, אלא את העותק של ערכו, הנמצא במשתנה אוטומטי.

נבהיר זאת על ידי הדוגמה שבתכנית הבאה:

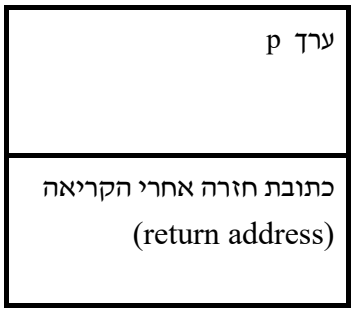
```
/* chapter 5, fourth example.
Example - print array */

#include <stdio.h>
void func (int *p)
{
   printf ("%d\n", *p);
   return;
}

main()
{
      int a[10];
      int *pa;

      a[0] = 8;
      pa = &a[0];
      func(a);
      func(pa);
```

```
}
```

תרגיל

יש להריץ את התכנית, ולראות את תוצאות שתי ההדפסות.

בשתי הקריאות לפונקציה func, יהיה מצב המחסנית **זהה**, ויכיל את ערכו של המשתנה האקטואלי: (המחסנית מתחילה בכתובת s1)



5.4 Address Arithmetic

שפת C מבטיחה כי אפס לא מהווה כתובת חוקית עבור נתונים. לשם בהירות, עדיף להקפיד ולהשתמש בקבוע NULL, המוגדר ב-<stdio.h>, כדי לציין ערך שאינו מצביע על "שום מקום".

אחת התכונות הבסיסיות של C היא העובדה שהאריתמטיקה של מצביעים נעשית בהתאם לטיפוסים, שעליהם הם מצביעים. החיבור והחיסור יבוצעו בהתאם לגודל הטיפוס, שעליו הם מצביעים. ההפרש בין שני מצביעים עוקבים במערך יהיה תמיד 1, ללא כל תלות בטיפוס.

תרגיל

יש להריץ את התכנית הבאה:

```
/* chapter 5, fourth example */
/* Example of pointer increments */
/* pointer values are printed in hex */

#include <stdio.h>
```

```
main()
{
   static double d_array[10];
   double *dp = &d_array[4];

   static float f_array[10];
   float *fp = &f_array[4];
   /* The usage of static isn't mandatory. The program will
   show values of the data segment instead of stack values */

   printf ("The size of double is %d bytes\n", sizeof(double));
   printf ("The size of float is %d bytes\n\n", sizeof(float));
   printf ("The value of dp before increment is %p\n, dp);
   dp = dp + 1; /* dp +=1 or dp++ are equivalent */
   printf ("The value of dp after increment is %p\n", dp);

   printf("\n");

   printf ("The value of fp before increment is %p\n", fp);
   fp = fp + 1; /* fp +=1 or fp++ are equivalent */
   printf ("The value of fp after increment is %p\n", fp);

}
```

בתכנית זו הוגדרו שני מצביעים לטיפוסי שונים של נתונים.
יש לשים לב בכמה השתנה ערכו של כל מצביע לאחר שהוספנו לו 1.

נחזור לתכניות alloc ו-afree שבספר (עמודים 102-101):
alloc היא פונקציה המחזירה מצביע ל-n התווים שאותם רוצים להקצות, afree היא פונקציה המשחררת את השטח המוצבע על ידי p.

ALLOCSIZE הוא גודל המקום הפנוי להקצאה.
allocbuf הוא המערך המשמש שטח להקצאה.

allocp הוא מצביע למקום הפנוי הבא ב-allocbuf.
נשים לב לזהירות שבה מבצעים מחברי הספר את הבדיקות של alloc ו-afree :

למשל, ב-afree התנאי

$$p < allocbuf + ALLOCSIZE$$

שקול (מבחינה אריתמטית) לתנאי

$$p - ALLOCSIZE < allocbuf$$

וב-alloc, התנאי

$$allocbuf + ALLOCSIZE - allocp >= n$$

שקול אריתמטית לתנאי

$$allocbuf + ALLOCSIZE >= allocp + n$$

אולם חישוב הביטויים השקולים עלולים להוציאנו מחוץ לגבולות המערך allocbuf, ולכן לא נוקטים בהם.

למשל, ברוב המקרים שבהם נקרא לפונקציה afree, הביטוי p-ALLOCSIZE ייתן ערך שהוא מחוץ לגבולות המערך. גם הביטוי allocp+n יחזיר ערך מחוץ לגבולות המערך. ברוב המחשבים ולגבי רוב המהדרים, הביטויים הנ"ל אכן שקולים, אולם השפה לא מבטיחה זאת, ומשום כך התכנית אינה ניידת (portable). יש לשים לב שאין מבצעים התייחסות אל התוכן של allocp+n (באמצעות האופרטור *). חישוב הערך כשלעצמו הוא הגורם לחוסר הניידות.

5.5 Character Pointers and Functions

בכל פעם שהמהדר נתקל במחרוזת קבועה, הוא מקצה מקום (בחלק ה-DATA של התכנית) בגודל השווה לאורך המחרוזת, ומקום נוסף עבור תו סיום המחרוזת '\0'. כל התייחסות למחרוזת היא למעשה התייחסות אל כתובת המקום הראשון בזיכרון, שממנה החלה ההקצאה.

הכתובת שממנה החלה ההקצאה היא זו המשמשת את המהדר בעת קידוד התכנית. למשל, אם המחרוזת היא פרמטר אקטואלי של פונקציה (כמו ב-printf), תועבר כתובת המחרוזת לפונקציה.

המהדר מאתחל את תאי הזיכרון במקום שהוקצה לערכי ה-ASCII של תווי המחרוזת.

אחת האופציות של המהדר גורמת ליצירת קובץ אסמבלי של התכנית, שעליה ביצענו הידור. ניתן להתבונן בקובץ, ולראות כיצד מקצה המהדר מקום למחרוזת, ומאתחל את ערכיה.

קטע הקוד הבא הוא חוקי לגמרי:

```
char target[100];
strcpy(target, "hello there");
```

זו הסיבה לכך ש-strcpy מצהירה על הפרמטרים שלה כעל *char.

יש לשים לב:
יש לקרוא ל-strcpy רק כאשר יודעים בוודאות שהיעד מכיל די מקום להעתקת המקור. אחרת נקבל גלישה.

תרגיל

מה יקרה אם נבצע:

```
strcpy("hello", "bye");
```

פתרון

נקבל שגיאת ריצה מכיוון שמנסים לשנות מחרוזת קבועה.

תרגיל

יש לבצע את תרגיל 3-5: יש לכתוב גרסה עם מצביעים של הפונקציה strcat, שאותה ראינו בפרק 2, עמוד 48: strcat(s,t) מעתיקה את המחרוזת t, לסופה של המחרוזת s.

פתרון (5-3)

```
/*  strcat: concatenate t to the end of s; s must be big enough */
char * strcat(char *s, char *t)
{
    char *keep = s;

    while (*s != '\0')
        *s++; /* find the end of s */

    while ((*s++ = *t++) != '\0')
        ; /* copy t */

    return(keep); /* To resemble the library */
}
```

תרגיל

יש לבצע את תרגיל 5-5: יש לכתוב גרסאות חדשות של פונקציות הספרייה strncat, strncpy, strncmp אשר מפעילות לכל היותר את n התווים של הארגומנטים שלהן.
למשל, strncpy(s, t, n) מעתיקה לכל היותר n תווים של t ל-s.

פתרון (5-5)

```
/* Copy at most n characters of string t to s; return s.
    Pad with '\0's if t has fewer than n characters */
char * strncpy(char *s, char *t, int n)
{
    char *keep = s;

    while (n-- > 0 && *t != '\0')
        *s++ = *t++;

    while (n-- > 0)
        *s++ = '\0';

    return(keep);
}
```

```
/* Concatenate at most n characters of string t to string s.
    Terminate s with '\0' ; return s. */
char strncat (char *s, char *t, int n)
{
    char *keep = s;

    strncpy (s + strlen(s), t, n);
    return (keep);
}
```

```
/* Compare at most n characters of string s to string t.
    Return < 0 if s<t , 0 if s==t, or > 0 if s>t. */
int strncmp (char *s, char *t, int n)
{
    for (; *s = = *t; s++, t++)
        if (*s = = '\0' | | –-n <= 0)
                return (0);

    return (*s – *t);
}
```

Pointer Arrays; Pointers to Pointer 5.6

תרגיל

יש לבצע תרגיל 5-7:

יש לכתוב מחדש את הפונקציה readlines (עמוד 109 בספר), כך שתאחסן שורות במערך המוגדר בתכנית, במקום על ידי הקריאה לפונקציה alloc.

פתרון (5-7)

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

#define MAXLINES 5000   /* max number of lines */
#define MAXLEN   1000     /* max length of an input line */
char *lineptr[MAXLINES];
char lines[MAXLINES][MAXLEN];

/* getline (page 29): read a line into s, return length */
int getline(char s[], int lim)
{
    int c, i;

    for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; i++)
      s[i] = c;
    if (c == '\n') {
      s[i++] = c;
    }
    s[i] = '\0';
    return i;
}

/* readlines (page 109): read input lines */
int readlines(char *lineptr[], int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];

    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
        if (nlines >= maxlines || (p = malloc(len)) == NULL)
            return -1;
```

```
        else {
                line[len - 1] = '\0';  /* delete newline */
                strcpy(p, line);
                lineptr[nlines++] = p;
               }
     return nlines;
}

/* readlines2 (new function): read input lines */
/* store lines in array "lines" */
int readlines2(char lines[][MAXLEN], int maxlines)
{
    int len, nlines;

    nlines = 0;
    while ((len = getline(lines[nlines], MAXLEN)) > 0)
        if (nlines >= maxlines)
           return -1;
        else
           lines[nlines++][len - 1] = '\0';  /* delete newline */
    return nlines;
}

int main(int argc, char *argv[])
{
    /* read into cache */
    readlines2(lines, MAXLINES);

    if (argc > 1 && *argv[1] == '2')
    {
       puts("readlines2()");
       readlines2(lines, MAXLINES);
    }
   else
   {
       puts("readlines()");
       readlines(lineptr, MAXLINES);
    }

   return 0;
}
```

5.7 Multi-dimensional Arrays

גם מערכים רב-ממדיים מוקצים כגוש רצוף בזיכרון. ל-daytab בדוגמה שבספר (עמוד 111), מוקצים 26 מקומות רצופים בזיכרון.
הסידור בגוש נעשה ב-row-major (סידור לפי שורות).

תרגיל

קטע הקוד הבא הוא חוקי לחלוטין. יש לנסות לחזות את תוצאותיו:

```
/* chapter 5, fifth example*/
/* Example of multi-dimensional array referencing */

#include <stdio.h>
main(void)
{
    static char daytab[2][13]=
    {
        {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
        {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
    };
    char *x = &daytab[0][0];
/* x now has the address of the first element of daytab, i.e. the beginning of
daytab */
/* Now we can look at x as the beginning of an array which has 2×13=26
entries */
    printf ("%d, %d\n", *(x+15), x[2]);
    x = &daytab[1][0];
/* now x has the address of the second row of daytab. We can now look at x
as the beginning of an array which has 13 entries.
*/
    printf ("%d\n, x[2]);

    x = &daytab[0][2];
/* We can now look at x as the start address of an
```

```
array which has 24 entries */
    printf ("%d\n, x[0], x[13]);
}
```

העובדה שהמערך הוא רציף מאפשרת לנו לכתוב פונקציות כלליות (generic).
לדוגמה, הפונקציה double_mat מכפילה את איברי המערך הנתון פי שניים, ללא תלות במספר הממדים שהוצהרו בעבורו. יש לשים לב כיצד מועברים הפרמטרים:

```
/* Example of a generic function */
/* double_mat is a generic function which doubles every
array entry is gets */

#include <stdio.h>

void double_mat (int *mat, int num_of_ent)
/* This function takes a matrix and doubles each element of it */
/* mat is a pointer to the beginning of the matrix
    num_of_ent is the number of entries the matrix has.
    The matrix can be of any dimension */

{
    for (; num_of_ent > 0; num_of_ent—)
        mat[num_of_ent – 1] *= 2;
}
main(void)
{
    int i, j, k;

    int mat1[2] = {1, 2};
    int mat2[2][3] =
    {
        {1, 2, 3},
        {4, 5, 6}
    };
```

```
    int mat3[2][3][4] =
    {
        {
                {1, 2, 3, 4},
                {5, 6, 7, 8},
                {9, 10, 11, 12}
        },
        {
                {13, 14, 15, 16},
                {17, 18, 19, 20},
                {21, 22, 23, 24}
        }
    };

/* Notice that the matrices are being passed by taking the address of their
first entry */
    double_mat (&mat1[0], 2);
    double_mat (&mat2[0][0], 6);
    double_mat (&mat3[0][0][0], 24);

    for (i=0; i < 2; i++)
        printf ("%d ", matf[i]);
    printf ("\n");

    for (i=0; i < 2; i++)
        for (j=0; j < 3; j++)
                printf ("%d ", mat2[i][j]);
        printf ("\n");
    for (i=0; i < 2; i++)
        for (j=0; j < 3; j++)
                for (k=0; k < 4; k++)
                        printf ("%d ", mat3[i][j][k]);
    printf ("\n");
}
```

Pointers vs. Multi-dimensional Arrays 5.9

התכניות הבאות מהוות דוגמאות להבדלים הדקים שבין מערך תווים לבין מצביע לתווים. הבדלים אלה אינם נהירים, בדרך כלל, למי שעושה את צעדיו הראשונים בשפה. יש לקרוא את הדוגמאות ולנסות לעמוד על ההבדל בין תכנית 1 לתכנית 2:

תכנית 1

```
/*
  Consider the following small C program.
  Path: is an array of pointers to characters,
  Which has been initialized
  p-path: is a pointer to path, i.e it is a
  pointer to pointers to characters

  The program scans the array path, and prints each
  character string at its turn.

*/
#include <stdio.h>
char *path[ ] = {

        "/usr/pgmr/blm/ct/page",
        "/usr/pgmr/junk",
        "/usr/lib/news/history",
        "/usr/bin/vi",
        "Testing Initialization",
        NULL
};

char **p-path = path ; /* pointer to path */
main( )
{
    while ( *p-path != NULL )
```

```
        printf ("%s\n", *p-path++ );
}
```

תכנית 2

```
/*
    This program defines the variable bad_address
    - which is a pointer to characters. One of the
    elements of the array of pointers to characters - path
    - is initialized with this variable.
*/

#include <stdio.h>
char *bad_address = "Testing Initialization";
char *path[ ] = {

        "/usr/pgmr/blm/ct/page",
        "/usr/pgmr/junk",
        "/usr/lib/news/history",
        "/usr/bin/vi",
        bad_address, /* This is an illegal initialization */
        NULL
};

char **p-path = path ; /* pointer to path */

main( )
{
    while ( *p-path != NULL )
        printf ("%s\n", *p-path++ );
}
```

בשתי התכניות ביצענו אתחול למערך path, אשר כל אחד מאיבריו הוא מצביע למחרוזת תווים. בתכנית 2 אתחלנו את אחד מאיברי המערך (החמישי) כמצביע למחרוזת תווים. במקרה זה קיבלנו שגיאת קומפילציה:

"Iillegal initialization"

נראה מהו ההסבר לכך:

בתכנית 1 הגדרנו מערך תווים שהוא ביטוי **קבוע**, גם כאשר הוא מומר למצביע. זאת מכיוון שהוא מצביע לכתובת קבועה בזיכרון, כבר לאחר ההידור, הקישור או הביצוע. לעומת זאת, מצביע לתווים הוא **משתנה**.

נרחיב מעט את ההסבר:

נתונות שתי ההגדרות הבאות:

א) char bad_adresss[ ] = "Testing Initialization"

ב) char * bad_adress ="Testing Initialization"

שתי הגדרות אלה אינן שקולות. הראשונה מגדירה מערך תווים ששמו "bad_address", היכול להכיל את המחרוזת "Testing Initialization" יחד עם ה- '\0' המסיים, והוא מאותחל עם מחרוזת זו. השנייה יוצרת מערך תווים דומה/זהה, אך **ללא שם**, ומגדירה בנוסף **מצביע** למערך זה, ששמו bad_address. לכן ההגדרה השנייה יוצרת משתנה, ואי אפשר לאתחל בעזרתו מערך של קבועים.

הסבר זה יכול להבהיר גם מדוע האתחולים הבאים לא יעבדו:

```
char *y = "1234";
char x[    ] = y;
```

הכוונה כאן הייתה לגרום ל-x להצביע על אותו מקום ש-y מצביע עליו, כלומר למערך התווים "1234". אך את המערך x אפשר לאתחל רק עם קבועים, וכפי שראינו - y הוא משתנה.

לעומת זאת, בסדר ההפוך לא תהיה בעיה:

```
char y [   ] = "1234"
char * x = y;
```

כי y הוא מצביע (קבוע) למערך התווים "1234", ולכן x, שהוא מצביע (משתנה) לתווים, יכול לקבל את ערכו.

תרגיל

האם מותר לכתוב משפט כגון:

```
printf ("%c\n", 4["hello"]);
```

פתרון

למרות שהדבר נראה מוזר, זהו תחביר חוקי לחלוטין. יודפס התו 'o' , מכיוון שהוא התו הרביעי במערך התווים הנתון.

5.10 Command-line Arguments

השימוש בארגומנטים של שורת פקודה -command line arguments- מאפשר להעביר פרמטרים אל התכנית מתוך תכנית הקליפה (shell) של מערכת ההפעלה.
מערכות הפעלה שונות ישתמשו בפקודות shell אחרות. העקרונות דומים למדי. פעולות אלו הן שקופות (transparent) לתכנית C. הגישה אל הארגומנטים של שורת הפקודה היא מתוך תכנית ה-C, ולכן היא תהיה זהה בכל מערכת הפעלה.
עם זאת, יש לזכור כי שורת פקודה עשויה לכלול גם חלקים, שלא יעברו כארגומנטים לתכנית, אלא יטופלו מראש על ידי ה-shell.
למשל, במערכת UNIX , במקרה הבא של redirection:

```
prog arg1 arg2 > filename
```

כאן יטופל החלק "filename >" על ידי ה-shell שינתב את הפלט הסטנדרטי של התכנית prog לקובץ filename, ואילו רק prog arg1 arg2 ייכנסו למבנה הנתונים argv.

תרגיל

יש לבצע את תרגיל 5-10:
יש לכתוב את התכנית expr, אשר מחשבת היפוך ביטוי לחישוב מתמטי, משורת הפקודה של המשתמש, כאשר כל אופרטור או אופרנד הוא ארגומנט נפרד. לדוגמה, הביטוי:

```
2 3 4 + *
```

ייצור:

```
2*(3+4).
```

יש להניח כי הקלט אינו מכיל מספרים ממשיים, והאופרטורים הם ארבע פעולות החשבון הבסיסיות.

פתרון (5-10)

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

#define STK_SIZE 20

static int stk_ptr = 0;
static stack[STK_SIZE];  /* stack and stack pointer */

void push (int val)
{
    if (stk_ptr >= STK_SIZE)
    {
        printf ("push: stack overflow\n");
        exit (1);
    }
    else
        stack[stk_ptr++] = val;
}
int pop(void)
{
    if (stk_ptr <= 0)
    {
        printf("pop: stack underflow\n");
        exit(2);
    }
    return(stack[−−std_ptr]);
}
```

```
main (int argc, char *argv[ ])
{
    int i;
    int tmp;
    if (argc = = 1) /* no arguments */
        return (0);

    for (i = 1; i < argc; i++)
        if (isdigit(*argv[i]))
                push (atoi(argv[i]));
        else
                switch(*argv[i])
                {
                case '+';
                        push (pop( ) + pop( ));
                break;

                case '*';
                        push (pop( ) * pop( ));
                break;

                case '–';
                        tmp = pop( );
                        push (pop( ) – tmp);
                break;

                case '/';
                        tmp = pop( );
                        push (pop( ) / tmp);
                break;

                default;
                        printf ("Unknown operator %c\n", *argv[i]);
                break;
```

```
            }
    printf ("%d\n", pop( ));
}
```

Pointers to Functions 5.11

השימוש במצביעים לפונקציות והעברתם של מצביעים אלה כפרמטרים מאפשרים כתיבת פונקציות כלליות (generic). בדוגמה שבסעיף זה מעבירים מצביע אל פונקציית ההשוואה, numcmp, הדרושה לתהליך המיון. פונקציית השוואה זו יכולה להיות כל פונקציה המקבלת שני מצביעים כפרמטרים ומחזירה שלם. יש לשים לב שאמרנו מצביעים בלי לציין על איזה טיפוס מסוים הם מצביעים. זאת כדי שהמצביעים לפונקציות, שאותם אנו מעבירים כפרמטרים, יוכלו להצביע על טיפוסים שונים. ייתכן, למשל, שאנו מעוניינים להשוות בין שתי רשומות (records) באמצעות בחינה של שדות מסוימים ברשומה. לשם כך הגדירו ב-ANSI-C מצביע כללי (generic pointer), כמצביע על טיפוס void (* void). ניתן לבצע השמה אל ומאת טיפוס * void לכל מצביע אחר, ללא איבוד אינפורמציה וללא צורך ב-cast.
אולם לא ניתן להפעיל את האופרטור * על מצביע מטיפוס * void. זאת מכיוון שתוך כדי הפעלת האופרטור * על מצביע, ניגשים לתוכנו (כלומר, לאובייקט שהמוצבע מצביע עליו), ובמקרה של void תוכן זה אינו מוגדר.

נשים לב לקריאה ל-qsort בפונקציה main. הפרמטר הרביעי שמועבר ל-qsort הוא המצביע לפונקציית ההשוואה. כאשר המהדר מגיע בקוד לשורה זו, הוא יודע כי numcmp ו-strcmp הוצהרו כפונקציות, המקבלות שני מצביעים ל-char ומחזירות int. (ההצהרה על numcmp כתובה במפורש, ואילו ההצהרה על strcmp כלולה בקובץ ה-header ב-<string.h>.)

מצד שני, qsort מצפה שהפרמטר הרביעי שלה יהיה מצביע לפונקציה המקבלת שני מצביעים ל-void ומחזירה int, ולכן מבוצע cast מפורש ((* void,* void)(int(*) על שתי הפונקציות.

הואיל והמהדר יודע שאלה הן פונקציות, אין צורך להשתמש באופרטור & כדי לקבל את כתובתן (strcmp &). המהדר יודיע שפעולה כזאת מיותרת (כפי שאין לוקחים את כתובת המערך, אלא פשוט מעבירים את שמו).

בגרסת השפה שקדמה להגדרת ANSI-C, שימש המצביע ל-char (char *) בתפקיד של void *. היה כמובן צורך לבצע cast מפורש אל טיפוס זה וממנו, עבור כל טיפוס השונה מ- char *.

נשים לב לדרך שבה קוראים לפונקציית ההשוואה (comp) בפונקציה qsort. תחילה מפעילים את אופרטור ה-* על comp (לכך דואגים הסוגריים), ולאחריו הארגומנטים בצורה:

```
(*comp)(v[i],v[left])
```

אולם, כפי שראינו עד עד עתה, הרי שב-C קוראים לפונקציה בצורה:

```
strcmp(v[i],v[left])
```

ללא כל שימוש באופרטור *. (נזכור כי ייתכן מצב שבו נעביר את strcmp בתור הפרמטר comp).

לגבי פונקציות ב-C אין הבדל בין comp* ל-comp, **כאשר comp הוא מצביע לפונקציה.**

את הקריאה ל-comp בפונקציה qsort ניתן לכתוב גם בצורה:

```
if (comp)(v[i],v[left]) < 0)
```

תרגיל

יש לבצע את תרגיל 5-14:

יש לשנות את תכנית המיון, כך שתטפל בדגל r– אשר יציין מיון בסדר יורד. יש לשים לב ש- r– יעבוד בתיאום עם הדגל n– המציין מיון בסדר נומרי, ולא לקסיקוגראפי.

בנוסף נשים לב כי הספרייה הסטנדרטית מכילה את הפונקציה qsort, לכן יש לקרוא לה בשם אחר בתכנית. אחרת, תכנית הקישור תודיע ש-qsort מוגדרת פעמיים.

פתרון (5-14)

```
/* This is the sort program of chapter 5 section 11 (page 119)*/
/* with minor modifications to support –r option */

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

#define MAXLINES 5000 /* max number of lines to be sorted */

char *lineptr[MAXLINES];  /* pointer to text lines */


int reverse = 0; /* 1 if reverse order is wanted */

int readlines (char *lineptr[ ], int nlines);
void writelines(char *lineptr[ ], int nlines);

void qqsort (void *lineptr[ ], int left, int right, int (*comp)(void *, void *));
int numcmp(char *, char *);

/* sort input lines */
main (int argc, char *argv[ ])
{
    int nlines;              /* number of input lines read */
    int numeric = 0;       /* 1 if numeric sort */

    int cnt;           /* counter that will pass over argv */

    for (cnt = 1; cnt < argc; cnt++)
        if (strcmp(argv[cnt], "–n") = = 0)
                numeric = 1;
        else
```

```
        if (strcmp(argv[cnt], "–r") = = 0)
                reverse = 1;
        else
        {
                printf ("Unknown argument: \"%s\"\n", argv[cnt]);
                exit (1);
        }
    if ((nlines = readlines (lineptr, MAXLINES)) >= 0)
    {
        qqsort ((void **) lineptr, 0, nlines–1,
        (int(*)(void) *,void *))(numeric?numcmp:strcmp));
        writelines(lineptr, nlines);
        return 0;
    }
    else
    {
        printf("input too big to sort\n");
        return 1;
    }
}

/* qqsort: sort v[left]… v[right] into increasing/decreasing order */
void qqsort (void *v[ ], int left, int right,
    int (*comp)(void *, void *))
{
    int i, last;
    void swap (void *v[ ], int, int);

    if (left >= right) /* do nothing if array contains */
        return;      /* fewer than two elements */
    swap(v, left, (left + right) / 2);
    last = left;
    for (i=left + 1; i <= right; i++)
        switch (reverse)
```

```
        {
                case 0: /* increasing order */
                        if (comp(v[i], v[left]) < 0)
                        swap (v, ++left, i);
                break;

                case 1:  /* reverse order */
                        if (comp(v[i], v[left]) > 0)
                        swap (v, ++last, i);
                break;

                default:
                        printf("Invalid value of %d for reverse\n", reverse);
                break;
        }
    swap(v, left, last);
    qqsort(v, left, last – 1, comp);
    qqsort(v, last + 1, right, comp);
}

#include <math.h>
/* numcmp: compare s1 and s2 numerically */
int numcmp(char *s1, char *s2)
{
    double v1, v2;

    v1 = atof(s1);
    v2 = atof(s2);
    if (v1 < v2)
        return –1;
    else if (v1 > v2)
        return 1;
    else
        return 0;
```

```
}

/* exchange two elements */
void swap (void *v[ ], int i, int j)
{
    void *temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}

#define MAXLEN 1000
int getline (char *, int);
void * malloc(size_t);
/* readlines: read input lines */
int readlines (char *lineptr[ ], int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];
    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
        if (nlines >= maxlines | |
                (p = malloc(len) = = NULL)
                return –1;
        else
        {
                line[len – 1] = '\0';
                strcpy (p, line);
                lineptr[nlines++] = p;
        }
    return nlines;
}
```

```
/* writelines: write output lines */
void writelines(char *lineptr[ ], int nlines)
{
    int i;

    for (i=0; i < nlines; i++)
        printf ("%s\n", lineptr[i]);
}

int getline (char s[ ], int lim)
{
    int c, i;
    i = 0;
    while (--lim > 0 && (c = getchar( )) != EOF && c != '\n')
        s[i++] = c;
    if (c = = '\n')
        s[i++] = c;
    s[i] = '\0';
    return i;
}
```

סעיף 5.12, המטפל בהצהרות מורכבות, הוא מסובך יותר מאשר בפרקים הקודמים. הסעיף מסתמך על ידע מתקדם: דקדוק, עצי גזירה, פיסוק (parsing). נושאים אלו נלמדים בקורס "קומפילציה".

ניתן לפסוח על קריאת סעיף 5.12 בספר.

להלן הסבר על הצהרות ב-C: כיצד יש להבין אותן וכיצד ניתן ליצור הצהרות חדשות, לישויות הדרושות לנו.

לפני שנעבור לדיון מפורט, יש לדעת כי ההצהרות המובאות בדוגמאות הבאות הן למעשה נדירות מאוד. רק לעתים רחוקות נתקלים בהצהרות כה מורכבות, אולם, ההצהרות הנ"ל הן חלק מן השפה, ועיון בהן תורם להבנתה. לכן מובא ההסבר שלהלן.

הצהרות ב-C הן לעתים קשות לקריאה ולהבנה, בעיקר כאשר מעורב בהן האופרטור *. הקושי נובע מהעובדה ש-* הוא אופרטור תחילי (prefix operator), המופיע לפני הארגומנט, בעוד שהאופרטורים האחרים, המופיעים בהצהרות, הם אופרטורים סופיים (postfix operators), המופיעים אחרי הארגומנט.

כאשר רוצים לפענח הצהרות מורכבות ב-C, כדאי לזכור כי:

א. תחביר ההצהרה ב-C על משתנה מסוים, דומה מאוד לתחביר המשמש אותנו לגישה אל ערך המשתנה.

זהו הבדל גדול בין C לבין שפות עיליות אחרות, כגון פסקל או עדה, שבהן משקף תחביר ההצהרה את מבנה הטיפוס שעליו מצהירים (array of וכו').

ב. מסקנה מ-א': בעת הקריאה עלינו להתחשב בסדר הקדימות של אופרטורי C המופיעים בהצהרות. (על פי הטבלה בעמוד 53 בספר). למשל, לאופרטור ה-* קדימות נמוכה יותר מזו של האופרטורים [ ] (גישה למערך) ו-( ) (קריאה לפונקציה), ולשני האחרונים קדימות זהה.

כדי להבין הצהרות מורכבות כגון:

```
int *(*(*x)[6])(int, int);
```

נשתמש בשיטה המפרקת את הביטוי המורכב, בהתאם לסדר הקדימויות של האופרטורים. יש לזכור (סעיף א' לעיל) כי הגישה אל ערכו של משתנה, דומה מאוד להצהרה עליו, ולכן מבוצע הפירוק בהתאם לסדר, שאותו מכתיבים האופרטורים.

כל שיש להכיר הוא את ארבעת המבנים הבאים:

תחביר C	הסבר	
int x	x: int;	(x הוא מסוג int)
int *x	x: pointer to int;	
int x[n]	x: array[0..n–1] of int;	
int x(arg1, arg2, …)	x: function with arguments (arg1, arg2, …) returning int	

הטיפוס int הוא לצורך הדגמה בלבד. שקילויות דומות ניתן למצוא עבור כל טיפוס.

כל טיפוסי הנתונים ב-C הם הרכבה של ארבעת המבנים דלעיל, שייקראו להלן **מבנים אלמנטריים**.

דוגמה

עתה נמצא את x מתוך ההצהרה המורכבת שהוזכרה לעיל:

int *(*(*x)[6])(int, int)

הפענוח נעשה על ידי פירוק הדרגתי של המבנה המורכב, בהתאם לקדימת האופרטורים, למבנה פחות ופחות מורכב.

כל המבנה המורכב הינו int (עקב ההצהרה):

((*x)[6])(int, int):int

(זהו המבנה האלמנטרי הראשון).

השלב הבא בפירוק הוא הסרת אופרטור ה-* מצד שמאל:

(*(*x)[6])(int, int): pointer to int

נסביר מדוע הסרנו את אופרטור ה-* ולא את אופרטור הפונקציה: יש לזכור שהגישה לערך המשתנה היא בהתאם לסדר הקדימות, והיות שו-* נמוכה יותר, היא תהיה הפעולה האחרונה שנבצע על מנת להגיע ל-int. הואיל ואנו מתחקים אחר הסדר ההפוך של הפעולות, הראשון שאנו מסירים אותו הוא האחרון לביצוע.
השלב הבא:

(*(*x)[6]): function with two int arguments
returning pointer to int

כאן הסרנו את הסוגריים הימניים המייצגים פונקציה, היות שבעת הערכת הביטוי היינו לוקחים תחילה את הביטוי שבתוך הסוגריים ([6](*x)*) (יש לזכור: לסוגריים קדימות גבוהה יותר מזו של סוגרי הפונקציה, כי האסוציאטיביות היא משמאל לימין).
השלב הבא הוא הורדת הסוגריים:

*(*x)[6]: function with two int arguments
returning pointer to int

השלב הבא הוא הסרת ה-* (קדימותו נמוכה מזו של [6]). נקבל:

(*x)[6]: pointer to function with two
int arguments returning pointer to int

השלב הבא הוא הסרת [6], שקדימותו נמוכה מזו של הסוגריים:

(*x): array[0..5] of pointer to
function with two int arguments
returning pointer to int

השלב הבא הוא הסרת הסוגריים. התיאור המילולי נשאר זהה:

*x: array[0..5] of pointer to
function with two int arguments
returning pointer to int

ולבסוף מתגלה x.

x: pointer to array [0..5] of
pointer to function with two
int arguments returning pointer to int

דוגמה נוספת

char(*(*z[3])(int))[5]

שלב 1:

(*(*z[3])(int))[5]: char

שלב 2:

(*(*z[3]))(int): array[0..4] of char

שלב 3:

(*z[3])(int): pointer to array[0..4] of char

שלב 4:

(*z[3]): function with one int argument returning pointer to array[0..4] of char

*z[3]: function with one int argument returning pointer to array[0..4] of char

z[3]: pointer to function with one int argument returning pointer to array [0..4] of char

z: array[0..2] of pointer to function with one int argument returning pointer to array[0..4] of char

השימוש במילת המפתח typedef, שעליה נלמד בפרק 6, מאפשר קריאות ובנייה קלה יותר של טיפוסי נתונים מורכבים.

הכיוון ההפוך: הצהרת הטיפוס מן התיאור המילולי שלו, הוא פשוט יותר. מתחילים בשם המשתנה ובונים סביבו את סדרת האופרטורים. יש רק לזכור כי בכל פעם שאנו רוצים להוסיף אופרטור ( ) או [ ], ובהצהרה החלקית שבנינו מופיע האופרטור * כאופרטור האחרון, יש להקיף את כל הביטוי החלקי שנבנה בסוגריים רגילים, כדי לתת לו קדימות גבוהה יותר.

לדוגמה, נניח כי אנו רוצים להצהיר על משתנה y כעל:

```
y:  function with one int argument returning
    pointer to array[0..6] of pointer to function
    with two char aruments returning char;
```

תחילה נבנה פונקציה עם משתנה int אחד:

```
pointer y(int):  pointer to array[0..6] of
                 to function with two char arguments
                 returning char
```

האופרטור הבא הוא *. באמצעות הוספת * מצד שמאל של הביטוי החלקי, נצהיר על y כעל פונקציה המחזירה מצביע:

```
*y(int):  array[0..6] of pointer
          to function with two char
          arguments returning char
```

האופרטור הבא שיש להוסיף הוא מערך. הואיל והאופרטור האחרון שצורף הוא *, יש לשים סוגריים על הביטוי הקודם כדי להבטיח את קדימותו בעת ההערכה (evaluation).

```
(*y(int))[7]:  pointer to function
               with two char arguments
               returning char.
```

האופרטור הבא הוא * (היות שנאמר pointer to):

```
*(*y(int))[7]:  function with two
                char arguments returning char;
```

שוב, יש להוסיף סוגריים על הביטוי החלקי כדי להגדיל את הקדימות היחסית לפונקציה. הפעם נעשה את שלב הסוגריים בנפרד, ליתר בהירות:

```
(*(*y(int))[7]):  function with two char
```

arguments returning char

ונעבור לפונקציה:

(*(*y(int))[7])(char,char): char

ובשלב האחרון אנו מגיעים להצהרה הסופית:

char (*(*y(int))[7])(char,char)

דוגמה נוספת

z: array[0..8] of pointers to function with one int argument that return pointer to an array[0..1] of pointers to int

z[9]: pointer to function with one int argument returning pointer to an array[0..1] of pointers to int

*z[9]: function with one int argument returning pointer to an array[0..1] of pointer to int

(*z[9])(int): pointer to an an array[0..1] of pointer to int

*(*z[9])(int): array[0..1] of pointer to int

(*(*z[9])(int))[2]: pointer to int

((*z[9])(int))[2]: int

ולבסוף

int *(*(*z[9])(int))[2]

נקודות לסיכום פרק 5

הנושאים העיקריים שנדונו בפרק זה הם:

- האופרטורים * ו-&.
- מצביע כפרמטר לפונקציה.
- הקשר בין מצביעים למערכים.
- אריתמטיקה של כתובות.
- הקצאה דינמית.
- מחרוזות ומצביעים.
- מערכים דו-ממדיים, וההבדל בינם לבין מערך של מצביעים.
- ארגומנטים של שורת פקודה.
- מצביעים לפונקציות.
- הבנת הצהרות מורכבות, דוגמאות הכוללות את האופרטור *.

תזכורת שהופיעה בפרק:

- רשומת הפעלה (activation record)

פרק 6 Structures

6.1 Basics of Structures

נשים לב למקרה של אתחול רשומה (structure): אם מספר האיברים באתחול קטן ממספר השדות ברשומה, האתחול נעשה לשדות הראשונים ברשומה, לפי הסדר שבו הוצהרו, וכל שאר השדות מאותחלים לאפס.

דוגמה

```
struct point {
    int x;
    int y;
    int z;
};

struct point pt = {320, 100};
```

במקרה זה השדה x יאותחל בערך 320, השדה y – בערך 100, ואילו השדה z יקבל את הערך 0.

הערות

אם מספר ערכי האתחול גדול ממספר השדות, מתקבלת מהמהדר הודעת שגיאה.
אין ב-C דרך קיצור לגישה אל שדות הרשומה.

6.2 Structures and Functions

בשפת C לא קיים אופרטור להשוואת רשומות. ניתן להשוות בין שדות, ואפשר להשוות בין רשומות באמצעות פונקציית הספרייה memcmp (עמוד 250 בספר).

דוגמה

x ו- y הן רשומות מסוג כלשהו. ניתן להשוות ביניהן על ידי:

memcmp (void) *)&x, (void *)&y, sizeof(x));

אם הרשומות זהות, פונקציית הספרייה תחזיר 0.

תרגיל

התכנית הבאה מגדירה נקודות `(x,y,z)` במרחב. יש לכתוב את הפונקציה is_equal המוצאת אם קיים שוויון בין הנקודות הנתונות. הפונקציה תחזיר 1 אם הנקודות שוות, ו-0 אם הן שונות. נשים לב כי הנקודות מוגדרות כמשתנים גלובליים.

```
#include <stdio.h>
typedef struct {
    float x;
    float y;
    float z;
} my_point;

my_point x1={0,1,10}, x2={0,1.1,90.5};

void main (void)
{
    if (is_equal())
        printf(" x1 is equal to x2 \n ");
    else
        printf("x1 is not equal to x2 \n  ");
}
```

הצעת פתרון

הפונקציה is_equal יכולה להכיל את השורה:

```
memcmp (void) *)&x1, (void *)&x2, sizeof(x1));
```

או, לחלופין, להשתמש בהשוואת שדות באופן הבא:

```
if ((x1.x == x2.x) && (x1.y == x2.y) && (x1.z == x2.z))
```

6.3 Array of Structures

נסביר את מבנה ההצהרה שבעמוד 132:

```
struct key {
        char *word;
        int count;
} keytab [NKEYS];
```

ההצהרה המופיעה עד סוף הסוגריים המסולסלים היא הצהרת טיפוס.
אין כאן הפרדה בין הצהרה על טיפוס של משתנה לבין הקצאת המקום. בהצהרה זו יש שילוב של שני אלמנטים אלו בו-זמנית: יש כאן הצהרה על טיפוס משתנה בשם key. בנוסף על כך, יש כאן הקצאת מקום למערך (בגודל NKEYS) של רשומות (structs) מטיפוס key, ששמו keytab.

נשים לב לשימוש בערכי המצביעים, בתהליך החיפוש של הפונקציה binsearch (עמוד 134):
המשתנה high מקבל בהתחלה ערך החורג באחד (הכוונה באלמנט אחד של המערך) מסוף המערך. מותר לעשות זאת (עמוד 103 למעלה). הדבר נעשה לצורך התנאי לעצירת הלולאה (while(low<high. שהרי אם האיבר שאותו מחפשים הוא האחרון במערך, אזי בסוף התהליך נקבל: low=&tab[n-1] ואין צורך להיכנס ללולאה.

6.4 Pointers to Structures

כפי שניתן להגדיר מצביע למשתנה רגיל, ניתן גם להגדיר משתנה מסוג "מצביע לרשומה".
לדוגמה:

```
struct key {
    char *word;
    int count;
};

struct key kt;
struct key *p_kt = &kt;
```

כאן מוגדר המבנה key (כפי שמופיע בעמוד 133 בספר) ואחריו שני משתנים: המשתנה kt הוא מסוג struct key, המשתנה p_kt הוא מסוג מצביע ל- struct key. כאן המשתנה p_kt מאותחל בכתובת של kt.
ניתן לפנות למשתנה פנימי של הרשומה שעליה מצביעים. לדוגמה, אם נרצה לגשת למשתנה הפנימי count של הרשומה שעליה מצביע p_kt, נרשום זאת כך:

```
(*p_kt).count
```

6.5 Self-referential Structures

פונקציית הספרייה malloc מאפשרת הקצאת זיכרון בשפת C. יש לשים לב: הפונקציה לא קובעת את גודל הזיכרון שיוקצה. יש לומר בפירוש מהו גודל הזיכרון המבוקש.
לדוגמה, בעמוד 142 מופיע:

```
malloc(sizeof(struct tnode))
```

כאן מוקצה מקום בגודל הרשומה tnode.

malloc מחזירה מצביע מטיפוס * void. מכאן שפעולות ה-cast ב-talloc וב-strdup (עמודים 142-143) הן למעשה מיותרות:

```
/* talloc:  make a tnode */
  struct tnode *talloc(void)
  {
     return (struct tnode *) malloc(sizeof(struct tnode));
  }
```

כאן הפונקציה talloc מחזירה ערך מסוג * struct tnode, כך שאין צורך בפעולת ה-cast המפורש בפונקציה. הדבר יתבצע גם באופן בלתי מפורש (implicit).

גם הפונקציה strdup מחזירה ערך מסוג ידוע: * char, כך שאין צורך בפעולת ה-cast המפורש:

```
char *strdup(char *s)   /* make a duplicate of s */
  {
     char *p;
```

```
    p = (char *) malloc(strlen(s)+1); /* +1 for '\0' */
    if (p != NULL)
       strcpy(p, s);
    return p;
}
```

בכל קריאה ל-malloc חשוב לבדוק את הערך ש-malloc החזירה. ערך שונה מ-NULL מהווה סימן לכך שההקצאה הצליחה (וכמובן מצביע למקום המוקצה).
פונקציית הספרייה free משחררת את הזיכרון, שהוקצה על ידי malloc. היא מקבלת כפרמטר מקום בזיכרון, שאותו קיבלנו מקריאה קודמת ל-malloc.

נדגיש כי יש חשיבות רבה לשחרור הזיכרון שהוקצה, לאחר גמר השימוש בו. לטעויות בשחרור הזיכרון, יש השלכה חמורה במיוחד בתכניות הפועלות מבלי להסתיים (רצות עד לבקשת סיום מפורשת). תופעת "זליגת" הזיכרון (איבוד מתמשך של זיכרון) מתגלה לעיתים רק בחלוף זמן רב, ותוצאותיה עלולות להשפיע גם על פעולתם של תהליכים אחרים, אשר יסבלו ממחסור במשאבי זיכרון.
נשים לב כי בעיות הנובעות מאי שחרור זיכרון עלולות להתגלות במקומות בלתי צפויים בתכנית. ייתכן, למשל, שלא נצליח להפעיל קטעים ופונקציות בתכנית, מכיוון שניסיון להקצות זיכרון נוסף נכשל, בשל מחסור בשטח זיכרון פנוי.

תרגיל

יש לבצע את תרגיל 5-13: נדרש לכתוב את הפונקציה tail, אשר מדפיסה את n השורות האחרונות מהקלט שלה. ברירת המחדל תהיה ש-n הוא 10, אולם יש לאפשר שינוי על ידי ארגומנט, כך שכתיבת tail –n תדפיס את n השורות האחרונות.
התכנית צריכה להתנהג באופן הגיוני, גם אם הקלט או הערך n אינם הגיוניים. יש לנצל את הזיכרון באופן היעיל ביותר, כלומר לאחסן את השורות באמצעות מצביעים, ולא על ידי מערך דו-ממדי בגודל קבוע.
ניתן להניח שאורך שורה לא עולה על 100 תווים. מספר השורות בקובץ אינו מוגבל ויכול להיות גבוה. אם n קטן ממספר השורות בקובץ, יש להדפיס את כולו. יש להיעזר בפונקציה getline לקריאת השורות, ולהשתמש ב-malloc כדי להקצות מקום ל-n השורות האחרונות.

פתרון (5-13)

```
/* tail –n: print the last n lines
the default of n is 10 if it hasn't been specified from the command line. It is
assumed that lines can be no longer than 100 characters. If the number of
input lines is less than n then print all of them.

method:
a.  Get the value of n from the command line
b.  Allocate n pointers to char *; these pointers will hold the pointers to the n
lines. They will be handled as a sort of list. The variable head will always
point to the first char * in the list.
c.  Let number-of-lines = 0;
d.  Read next line.
e.  If list isn't full (number-of-lines read is less then n) then allocate a new
line. Copy to it the line which has been read and put this line in the list.
    Otherwise
        switch the oldest line (the one which is pointed to by head) with the
new line, and advance head to the next line.
f.  When there are no more lines to be read do the printing.
if number-of-lines read is less than n, print the

    first number-of-lines in the list
    otherwise
    print the lines from head to n and then the lines from 0 to head – 1.
*/



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

#define MAX_LINE_LEN 100
#define DEFAULT_N 10
```

```
/* getline: get line into s, return lingth */
int getline (char s[ ], int lim)
{
    int c, i;

    i = 0;
    while (––lim > 0 && (c = getchar( )) !+ EOF && c != '\n')
        s[i++] = c;
    if (c = = '\n')
        s[i++] = c;
    s[i] = '\0';
    return i;
}

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

    int number_of_lines = 0;
/* The number of lines that has been read */
    char **keep;
/* will hold the first address of the list returned by malloc */

    char **head;
/* will hold a pointer to the oldest line that has been read */
    char **tmp;
/* for indexing through the list while printing */

    char line[MAX_LINE_LEN];
    int tail = DEFAULST_N;

    if (argc > 1 && argv[1][0] = = '–')
        tail = atop(&argv[1][1]);
```

```
    head = keep = malloc (tail * sizeof(char *));
/* This will allocate a block of tail char * */
    if (head = = NULL)
    {
        printf("malloc failed\n");
        exit (1);
    }
    while (getline(line, MAX_LINE_LEN) > 0)
    {
        if (number_of_lines < tail)

                /* less than tail lines have been allocated */
        {
                *head = malloc (MAX_LINE_LEN);
                if (*head = = NULL)
                {
                        printf("malloc failed\n");
                        exit (1);
                }
        }

        number_of_lines++;
        strcpy (*head, line);
        head++;
        if (head - keep = = tail)
                head = keep;
/* back to the beginning of list */
    }

    if (number_of_lines >= tail)
        for (tmp = head; tmp - keep < tail; tmp++)
                printf("%s", *tmp);
/* Printing from head (which always points to the oldest line) to the end of
the list */
```

```
    for (tmp = keep; tmp != jead; tmp++)
        printf("%s", *tmp);
/* Printing the rest. After we printed head[9] (default case) We start printing
from the beginning till one entry before head */
}
```

תרגיל

יש לבצע את תרגיל 6-3: נדרש לכתוב תכנית (cross-referencer), אשר תדפיס רשימה של כל המילים במסמך, ולכל מילה – רשימה של מספרי השורות שבהן היא מופיעה. אין צורך להתייחס למילים שכיחות כגון and, the.

בפתרון נשתמש בעץ בינארי כמבנה הנתונים. כמו כן, ניעזר בפונקציות להוספה ולחיפוש, אשר הוצגו בסעיף זה, ונתאימן לבעיה זו.

פתרון (6-3)

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

struct lunes_lust
{
    int line_num;
    struct lines_list *next;
};

struct tnode
{
    char *word
    int count;
    struct tnode *left;
```

```
    struct tnode *right;
    struct lines_list *head;
};


/* The above pointer will be the head of a list of integers that will
hold the line numbers in which the word appears */

#define BUFSIZE 100
static char (buf[BUFSIZE];
static int bufp = 0;

int my_getch(void)
{
    return (bufp > 0 ? buf[--bufp] : getchar( ));
}

void my_ungetch (int c)
{
    if (bufp >= BUFSIZE)
        printf ("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

#define MAXWORD 100

/* A modified version of getword; This version returns in it's
argument the line number of the word ot read */

int getword(char *word, int lim, int *line_num)
{
    int c;
    static line = 1;
```

```
    char *w = word;

    while (isspace(c = my_getch( )))
        if (c = = '\n')
                line++;
    *line_num = line;

    if (c ~= EOF)
        *w++ = c;

    if (!isalpha(c))
    {
        *w = '\0';
        my_ungetch (c);
                /* It might be \n so better keep it */
        return c;
    }

    for (; --lim > 0; w++)
        if (!isalnum (*w = my_getch( )))
        {
                my_ungetch (*w);
                break;
        }
    *w = '\0';
    return word [0];
}

/* new_line_ref: allocate space for lines list record */
struct lines_list *new_line_ref(void)
{
    struct lines_list *tmp;

    tmp = malloc (sizeof(struct lines_list));
```

```
    if (tmp = = NULL)
    {
        printf("new_line_ref: malloc faided!\n");
        exit (1);
    }
    return (tmp);
}

/* treeprint: in-order of tree p */
void treeprint(struct tnode *p)
{
    struct lines_list *tmp;

    if (p != NULL)
    {
        treeprint (p->left);
        printf ("%4d %s on lines ", p->word);
        for (tmp = p->head; tep != NULL; tmp = tmp -> next)
                printf("%d ", tmp->line_num);
        printf("\n");
        treeprint (p->right);
    }
}
/* talloc: make a tnode */
struct tnode *alloc(void)
{
    struct tnode *tmp;

    tmp = malloc(sizeof(struct tnode));
    if (tmp = = NULL)
    {
        printf("talloc: malloc failed!\n");
        exit (1);
    }
```

```
    return tmp;
}
/* addtree: add a node with w on line 1, at or below p */
struct tnode *addtree (struct tnode *p, char *w, int 1)
{
    int cond;
    struct lines_list *tmp;

    if (p = = NULL)
    {
        p = talloc( );
        p->word = strdup(w);
        p->count = 1;
        p->left = p ->right = NULL;
        p->head = new_line_ref ( );
/* A new word; start a nes list for it */
        p->head->line_num = 1;
        p->head->next = NULL;
    }
    else if ((cond = strcmp (w, p->word)) = = 0)
    {
        p->count++;
        tmp = new_line_ref ( );
        tmp->line_num = 1;
        tmp->next = p->head;
        p->head = tmp;
    }
    else if (cond < 0)
        p->left = addtree(p->left, w, 1);
    else
        p->right = addtree (p->right,w,1);
    return (p);
}
```

```
/* Word frequency count with line numbers */
main( )
{

    int cur_line;
    struct tnode *root;

    char word[MAXWORD];

    root = NULL;
    while (getword(word, MAXWORD, &cur_line != EOF)
        if (isalpha(word[0])
                root = addtree (root, word, cur_line);
    treeprint(root);
    return 0;
}
```

Table Lookup 6.6

תרגיל

יש לבצע את תרגיל 6-5: נדרש לכתוב את הפונקציה undef, אשר מסירה שם והגדרה מתוך טבלה, המתוחזקת על ידי הפונקציות lookup ו-install.

יש לזכור: strdup קוראת באופן עקיף ל-malloc, לכן יש לשחרר את הזיכרון שהוקצה עבור השדות name ו-defn, לפני שמשחררים את הרשימה עצמה. הפרמטר ל-undef יכול להיות גם מחרוזת, שלא הוגדרה כלל על ידי defn. במקרה זה לא יבוצע דבר.

פתרון (6-5)

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

```
#define HASSIZE 101

struct nlist
{
    char *name;  /* defined name */
    char *defn;   /* replacement text */
    struct nlist *next;  /* next entry in chain */
};

static struct nlist *hashtab[HASHSIZE];
/* pointers table */

/* hash: form hash value for string */
unsigned hash (char *s)
{
    unsigned hashval;


    for (hashval = 0; *s != '\0'; s++)
        hashval = *s + 31 * hashval;
    return hashval % HASHSIZE;
}

/* lookup: look for s in hashtab */
struct nlist *lookup (char *s)
{
    struct nlist *np;
    for (np=hashtab[hash(s)]; np != NULL; np = np -> next)
        if (strcmp (s, np->name) = = 0)
                return np;
    return NULL;
}

/* install: put (name, defn) in hastab */
struct nlist *install(char *name, char *defn)
{
    struct nlist *np;
    unsigned int hashval;

    if ((np = lookup (name)) = = NULL) /* Not found */
    {
        np = malloc (sizeof(*np));
/* it is also possible to write sizeof (struct nlist) */
        if (np = = NULL | |
```

```
            (np->name = strdup(name)) = = NULL)

/* notice that the above line relies on the order in which conditional
expressions are avaluated. if np = = NULL then no reference to
NULL pointer will be made */
        {
            printf ("malloc: failed in install\n");
            exit (2);
        }
        hashval = hash(name);
        np->next = hashtab[hashval];
    }
    else /* already there */
        free((void *) np->defn);
        /* Allocated by strdup */
    if ((np->defn = strdup (defn)) = = NULL
    {
        printf("install: failed on strdup\n");
        exit (3);
    }
    return np;
}
/* undef: undefined name */
void undef (char *name)
{
/* there is one small problem. In order to free a record from the list
we must know who is it's predecessor in the list. It also may not have
any predecessor (if it is the first on the list) */

    struct nlist *current, *pred;
    int hashval = hash(name);

    current = NULL;

    if (hashtab[hashval] != NULL &&
        (strcmp(hashtab[hashval]->name, name = = 0))
    {
        current = hashtab[hashval];
        hashtab[hashval] = hashtab[hashval]->next;
        free ((void *) current->name);
        free ((void *) current->defn);
        free ((void *) current);
        return;
    }
    else if (hashtab[hashval] != NULL)
```

```
    {
        pred = hashtab[hashval];
        for (current = pred->next; current != NULL;
                pred = current, current = current->next)
                if (strcmp(current->name, name) = = 0)
                        break;
    }

    if current != NULL)
    {
        pred=current->next;
        free ((void *) current->name);
        free ((void *) current->defn);
        free ((void *) current);
        return;
    }
    return;
}

main( )
{
    struct nlist *tmp;
    int i;

    for (i = 0; < HASHSIZE; i++)
        hashtab[i] = NULL;

    hashtab[hash("X")] = install ("X", "1");
    hashtab[hash("BLABLA")] = install ("BLABLA", blue");
    hashtab[hash("hello")] = install ("hello", there");

    tmp = lookup ("hello");
    if (tmp != NULL)
        printf ("%s is replaced by %s\n", tmp->name,
                tmp->defn);
    tmp = lookup ("BLABLA");
    if (tmp != NULL)
        printf ("%s is replaced by %s\n", tmp->name,
                tmp->defn);
    tmp = lookup ("X");
    if (tmp != NULL)
        printf ("%s is replaced by %s\n", tmp->name,
                tmp->defn);

    undef ("X");
```

```
    undef("not-here");
    undef("hello");
    tmp = lookup ("hello");
    if (tmp != NULL)
        printf ("%s is replaced by %s\n", tmp->name,
                tmp->defn);
    tmp = lookup ("BLABLA");
    if (tmp != NULL)
        printf ("%s is replaced by %s\n", tmp->name,
                tmp->defn);
    tmp = lookup ("X");
    if (tmp != NULL)
        printf ("%s is replaced by %s\n", tmp->name,
                tmp->defn);
}
```

6.7 Typedef

נדגיש כי ההגדרה typedef לא יוצרת סוג (type) חדש, אלא מוסיפה שם לסוג קיים. אחת הסיבות העיקריות לשימוש ב-typedef היא בהירות התוכנה. הדבר משמש כתיעוד טוב של התוכנה, ומונע חוסר בהירות, ואף טעויות כגון חיבור שני סוגים שונים. לדוגמה, כך נוכל להימנע מחיבור של מספר "התפוחים" עם מספר "התפוזים", אם הגדרנו אותם כסוגים שונים (ולא כסוג משותף, כגון "פרי").

דוגמה

נראה כיצד להשתמש ב-typedef כדי להגדיר טיפוס נתונים חדש, שהוא מערך בן שלושה איברים שכל אחד מהם מטיפוס int. שם הטיפוס החדש יהיה array_3int.

פתרון

typedef int array_3int [3];

אילו רצינו להקצות מקום למערך בשם ar3 בן שלושה איברים, ללא הגדרת typedef, היינו רושמים:

int ar3 [3];

בעוד שבעזרת הטיפוס החדש שהגדרנו, נוכל לבצע את אותה הקצאת מקום בצורה:

array_3int ar3;

תרגיל

א. באמצעות typedef, יש להגדיר טיפוס complex שייצג מספר מרוכב, בעל חלק דמיוני וחלק ממשי.

ב. יש לכתוב פונקציה המחברת שני מספרים מרוכבים. הפונקציה תקבל שני מצביעים למשתנים מטיפוס complex ותחזיר משתנה מטיפוס complex.

ג. כמו ב', אך יוחזר מצביע לטיפוס complex.

פתרון

```
/* first part */
typdef struct complex
{
    double real;
    double image;
} complex;

/* second part */
complex add_c (complex *x, complex *y)
{
    complex tmp;

    tmp.real = x->real + y->real;
    tmp.imag = x->imag + y->imag;
    return (tmp);
}

/* third part */
/* WRONG version; replace tmp with tmp *  */
complex *add_c2 (complex *x, complex *y)
{
    complex tmp;
```

```
    tmp.real = x->real + y->real;
    tmp.imag = x.imag + y.imag;
    return (&tmp);
}

/* Think for a while: why the above version is wrong? */

/* Correct version */
complex *add_c3 (complex *x, complex *y)
{
    complex *tmp;

    tmp = malloc (sizeof(complex));
/* malloc's returned value was not checked, but must be done nevertheless
*/
    tmp.real = x->real + y->real;
    tmp.imag = x->imag + y->imag;
    return (tmp);
}

/* The third part version was wrong because it returned a value which was
allocated on the stack, therefore, would not exist after return from the
function */
```

6.8 Unions

השימוש ב-union נוח כאשר רוצים לאחסן נתונים, מסוגים שונים, באותו המשתנה ובאותו שטח זיכרון. החיסרון הוא חוסר האפשרות לבדוק נכונות, באמצעות טיפוסים, כפי שראינו בסעיף קודם.

לדוגמה, גילאי ילדים ניתן לייצג על ידי משתנה מסוג float, המכיל את הגיל, או בשרשרת תווית המכילה את תאריך הלידה של הילד. על ידי union נוכל לאחסן את הייצוג הנתון לנו, באותו הזיכרון:

```
union
{
        char date[9];
        float age;
} child;
```

המשתנה child יהיה גדול מספיק כדי להכיל את הגדול מבין שני הסוגים. גודלו של המשתנה הוא תלוי מערכת, ואינו מוכתב על ידי ANSI-C.

כאשר נרצה לאתחל באמצעות תאריך הלידה של הילד, נבצע:

```
union
{
        char date[9];
        float age;
} child = {"21.06.99"};
```

כאשר נרצה לאתחל באמצעות גיל הילד, נבצע:

```
union
{
        char date[9];
        float age;
} child = {.age = 6.5};
```

נשים לב: המטרה כאן היא להשתמש בכל פעם **באחד** משני הייצוגים, ולא בשניהם. כמו כן, ניתן לקבל פרשנות שונה לאותו מבנה.

6.9 Bit-fields

ניתן להגדיר שדה ביטים (bit field) בתוך רשומה או union. ההגדרה מאפשרת שימוש בביטים **בודדים** של שדה. היתרונות בשימוש זה הם חיסכון בזיכרון, ופעולות מהירות ביותר.

דוגמה

מטרת התכנית הבאה היא לבדוק ולהדפיס האם ביט הוא OFF או ON.

בשל האתחול במקרה זה (בשורה הראשונה של ה-main), אמורים לקבל את ההדפסה "bit is ON".

```
#include <stdio.h>

    struct {
                unsigned bit:1;
            } rank;

    int find_rank()
      {
       if ( rank.bit > 0 )
           return 1;
       else
           return 0;
      }

    int main()
      {
       rank.bit = 1;

       if ( find_rank() )
           printf( "bit is ON\n" );
       else
          printf( "bit is OFF\n" );

       return 0;
      }
```

נקודות לסיכום פרק 6

הנושאים העיקריים שנדונו בפרק זה הם:

- רשומה ואתחולה.
- שימוש ברשומות ובפונקציות.
- מערך רשומות.
- מצביע לרשומה.
- typedef.
- unions.
- bit fields.

תרגיל

יש להציג את המקרה המתאים ביותר לשימוש, עבור כל אחד ממבני הנתונים ברשימה הנ"ל.

פרק 7. Input and Output

אמצעי קלט ופלט כגון קבצים חיצוניים, תנועות עכבר וסרטי וידאו, אינם חלק משפת C. אולם השפה מאפשרת אינטראקציה (פעילות הדדית) עם אמצעים אלו. כל קלט או פלט מכל סוג שהוא (כולל וידאו, קול וכו') ניתן לראות כרצף של תווים או מספרים. כך ניתן להתייחס לכל אמצעי קלט/פלט, בעזרת אותן פעולות המשמשות לכתיבה/קריאה מקובץ. שפת C והספריות הסטנדרטיות משתמשות בתכונה זו, לשם הכללת השימוש באמצעי קלט/פלט.

זרם המידע (stream) הוא הרחבה של המושג קובץ (או קלט/פלט אחר). זרם תווי (text stream), למשל, מכיל שורות, וכל שורה מסתיימת בתו סוף-שורה (new line). אורך הקובץ/המידע אינו ידוע מראש, כלומר סופו אינו מוכתב.

כל אלו מספקים גמישות, המאפשרת לשפת C ולספריות הסטנדרטיות להתייחס לכל אמצעי קלט/פלט בצורה דומה, ללא תלות במימוש הפיזי.

7.1 Standard Input and Output

תרגיל

יש לבצע את תרגיל 7-1 שבספר:

יש לכתוב תכנית ההופכת אותיות קטנות (lower case) לאותיות גדולות (upper case), או אותיות גדולות לאותיות קטנות, על פי השם המופיע ב- argv[0] .

פתרון (7-1)

```
/* Converts input to upper case
   (if argv[0] begins with U or u) or lower case.
 */

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

```
int main(int argc, char **argv)
{
 int (*convcase[2])(int) = {toupper, tolower};
 int func;
 int result = EXIT_SUCCESS;

 int ch;

 if(argc > 0)
 {
  if (toupper((unsigned char)argv[0][0]) == 'U')
     func = 0;
  else
     func = 1;

  while ((ch = getchar()) != EOF)
  {
   ch = (*convcase[func])((unsigned char)ch);
   putchar(ch);
  }
 }
 else
 {
  fprintf (stderr, "Unknown name.\n");
  result = EXIT_FAILURE;
 }

 return (result);
}
```

7.2 Formatted Output – Printf

נשים לב שניתן להדפיס ערכים של מצביעים באמצעות %p.

ראינו כבר ש-printf לא דואג להתאמה בין מספר האיברים, המועברים להדפסה, לבין מספר התווים המתואר במחרוזת הפורמט. כמו כן, טיפוסים שאינם מתאימים למחרוזת הפורמט, גורמים להדפסת תוצאות שגויות.

דוגמה

התכנית הבאה אמורה להדפיס במדויק את סכום איברי המערך a, אולם לא נקבל את ההדפסה שרצינו (תרגיל והסבר בהמשך).

```
#include <stdio.h>
 #define LEN 3

   void print_sum( double a[], int n )
     {
        int i;
        double sum = 0;
        for( i = 0; i < n; i++ ) sum += a[i];
        printf( "Sum = %d\n", sum );
     }

   int main()
     {
       double x[] = { 1. 1, 2.0, 3.0 };

       print_sum( x, LEN );
       return 0;
     }
```

תרגיל

יש להריץ את הדוגמה ולראות את ההדפסה המתקבלת.

פתרון

בתכנית מופיע printf("Sum = %d\n", sum) , ולכן יודפס הסכום 0 ולא 6.0. יש לשנות ל: printf("Sum = %f\n", sum) , מכיוון שהפורמט צריך להתאים ל-double.

7.3 Variable-length Argument Lists

אם נתונה רשימת ארגומנטים לפונקציה, כאשר אורכה וסוגי הארגומנטים שבה אינם ידועים, ניתן להשתמש בקובץ ה-header הסטנדרטי <stdarg.h> (שבו משתמשים, למשל, בעמוד 156 בספר). קובץ זה הוא חידוש של ANSI-C.

חשוב להבין שקטעי המקרו המובאים בקובץ זה מאפשרים לעבור על הפרמטרים, שהוכנסו למחסנית זמן-הריצה, בעת הקריאה לפונקציה. אין צורך לדעת מהו גודל הזיכרון, השמור לכל טיפוס: קטעי המקרו דואגים לקדם את ערכי המצביע לפי הטיפוס שבו אנו מעוניינים.

דוגמה

חישוב סכום על ידי פונקציה בת מספר משתנה של פרמטרים.

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

/* compute sum using variable-length argument list */
void sum(char *, int, ...);

int main(void)
{
  sum("The sum of 1+2+3 is %d.\n",3,1,2,3);
  return 0;
}

void sum(char *string, int num_args, ...)
{
  int sum=0;
  va_list ap;
```

```
  int i;

  va_start(ap,num_args);
  for(i=0;i<num_args;i++)
    sum+=va_arg(ap,int);

  printf(string,sum);
  va_end(ap);
}
```

Formatted Input - Scanf 7.4

תרגיל

יש לבצע את תרגיל 7-4 :

יש לכתוב גרסה פרטית של הפונקציה scanf, עבור מספרים שלמים (int) וממשיים (float). יש להשתמש ב-scanf לקריאת הנתון כפי ש-minprintf משתמשת ב-printf (בעמוד 156 בספר).

פתרון (7-4)

```
#include <stdio.h>
#include <stdarg.h>
/* miniscanf: minimal scanf with variable argument list */
void mini_scanf (char *fmt, …)
{
    va_list ap;
    char *p;
    int *iptr;
    float *fptr;

    va_start (ap, fmt);
    for (p = fmt; *p != '\0'; p++)
    {
```

```
        if (*p != '%')
                continue;
        switch (*++p)
        {
        case 'd':
                iptr = va_arg(ap, int *);
                scanf("%d", iptr);
                break;
        case 'f':
                fptr = va_arg(ap, float *);
                scanf("%f", fptr);
                break;
        default:
                printf("mini_scanf: unknown format %c\n", *p);
                break;
        }
    }
    va_end (ap);
}

main( )
{
    int i;
    float f;

    mini_scanf("%f %d, &f, &i);
    printf("%f %d\n", f, i);
}
```

File Access 7.5

סוגי הגישה לקובץ הם:

r	קריאה בלבד (מקובץ קיים).
w	כתיבה בלבד. אם הקובץ קיים, הוא יימחק.
a	הוספה לקובץ קיים. אם לא קיים, ייווצר קובץ חדש (כך פעולת ההוספה שקולה ליצירת קובץ חדש).
r+	קריאה וכתיבה לקובץ קיים.
w+	קריאה וכתיבה לקובץ. אם הקובץ קיים, הוא יימחק. (מדובר בשינוי באמצע הקובץ. שינוי זה דורש, למעשה, העתקה של הקובץ כולו, ושמירתו מחדש).
a+	הוספה לקובץ ואפשרות קריאה ממנו.

בשפת C משתמשים במצביע לקובץ (file pointer), אבל ניתן להצביע לכל מקום בקובץ. לפיכך, יש צורך לעקוב אחר מיקום המידע בקובץ. נראה זאת, למשל, בדוגמה הבאה:

```
#include <stdio.h>
#define NUM 15

/* point record description */
struct point
{
   int x,y,z;
};

/* writes and then reads NUM records
  from the file "points_file". */
int main()
{
   int i,j;
   FILE *f;
   struct point pnt;

   /* create a file of NUM records */
   f=fopen("points_file","w");
   if (!f)
```

```
    return 1;
for (i=1;i<=NUM; i++)
{
    pnt.x=i;
    fwrite(&pnt,sizeof(struct point),1,f);
}
fclose(f);

/* read the NUM records */
f=fopen("points_file","r");
if (!f)
    return 1;
for (i=1;i<=NUM; i++)
{
    fread(&pnt,sizeof(struct point),1,f);
    printf("%d\n",pnt.x);
}
fclose(f);
printf("\n");

/* use fseek to read the NUM records
   in reverse order */
f=fopen("points_file","r");
if (!f)
    return 1;
for (i=NUM-1; i>=0; i--)
{
    fseek(f,sizeof(struct point)*i,SEEK_SET);
    fread(&pnt,sizeof(struct point),1,f);
    printf("%d\n",pnt.x);
}
fclose(f);
printf("\n");
```

```
    return 0;
}
```

7.6 Error Handling – Stderr and Exit

נדגיש כי יש להבדיל בין פונקציות ספרייה לוגיות, כגון strcpy, לבין פונקציות ספרייה, כמו fopen, התלויות במערכת ההפעלה, ולכן תוצאות פעולתן עלולות להשתנות מדי פעם.

מרבית פונקציות הספרייה ב-C מחזירות ערך, שממנו ניתן ללמוד אם הקריאה הצליחה. דוגמאות של פונקציות כאלה הן scanf, fopen, malloc. פונקציות אלו תלויות בפעולה תקינה של מערכת ההפעלה, ולא רק בלוגיקה של התכנית. לכן הכרחי לבדוק את תוצאות הפעלתן. יש להקפיד לבדוק אם הערך שחזר מלמד על הצלחה בקריאה לפונקציה. זה יחסוך זמן ניפוי (debugging) רב.

פונקציות ספרייה רבות, בעיקר אלה המנהלות קלט/פלט (fopen למשל) וזיכרון (malloc), זקוקות לשירותי מערכת ההפעלה לצורך מימושן. לעיתים חשוב לדעת מדוע נכשלה קריאה לפונקציית ספרייה. למשל, יכולות להיות מספר סיבות לכך ש-fopen החזירה NULL: קובץ לא קיים, היעדר הרשאת גישה לקובץ, מספר הקבצים הפתוחים גדול מדי וכדומה.

שפת ANSI-C מאפשרת לנו לדעת, ביתר דיוק, מדוע פונקציית ספרייה מסוימת נכשלה, באמצעות פונקציית הספרייה perror. הפונקציה perror מקבלת כפרמטר מצביע למחרוזת, ומדפיסה, בקובץ השגיאה הסטנדרטי (stderr), את המחרוזת שהעברנו כפרמטר, ואחריה שורה קצרה, המציינת, ביתר פירוט, את סיבת השגיאה (אם אכן הייתה שגיאה). לכן יש להשתמש תמיד ב-perror, בזמן פיתוח תוכנה.

דוגמה

נכתוב ונריץ את התכנית הבאה.

```
/* Example of perror */
# include <stdio.h>
main ( )
{
   FILE *fp;
```

```
    fp = fopen ("Hello there.bye", "r");
       /* There can't be a file with such a name */
    if (fp = = NULL)
    {
       perror ("myprog");
       exit (7);
    }
}
```

מידע על האופן המדויק שבו עובד perror ניתן למצוא ב-users guide (בסביבת UNIX על ידי כתיבת man perror).

7.7 Line Input and Output

הפונקציה fgets היא פונקציה שימושית, שבה ידוע המספר המקסימלי של תווים בשורה.

תרגיל

יש לבצע את תרגיל 7-6: נדרש לכתוב תכנית להשוואת שני קבצים, כאשר התכנית מדפיסה את ההבדל הראשון ביניהם.

על התכנית לקרוא את שמות הקבצים משורת הפקודה (command line). ניתן להניח כי אורך שורה אינו עולה על 80 תווים.

פתרון (7-6)

```
/* Compare two files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

FILE *my_open (char *name, char *mode)
{
   FILE *fp;
```

```
    char msg[30];

    fp = fopen (name, mode);
    if (fp = = NULL)
    {
        sprintf(msg, "%s", name);
        perror(msg);
        exit (1);
    }
    return (fp);
}

main (int argc, char *argv[ ])
{
    FILE *file1, file2;
    int line;
    char buf1[100], buf2[100];

    if (argc != 3)
    {
        fprintf(stderr, "Usage: %s file file\n", argv[0]);
        exit (2);
    }

    file1 = my_open (argv[1], "r");
    file2 = my_open (argv[2], "r");
    line = 1;
    while (fgets (buf1, sizeof(buf1), file1) != &&
        fgets (buf2, sizeof(buf2), file2) != NULL)
    {
        if (strcmp (buf1, fub2) != 0)
        {
                printf ("files differ: line %d\n%s: %s%s: %s",
                        line, argv[1], buf1, argv[2], buf2);
```

```
            exit (3);
        }
        line++;
    }
    fclose(file1);
    fclose(file2);
}
```

7.8 Miscellaneous Functions

בפרק זה ובנספח B בספר מתוארות פונקציות שימושיות בנושאים שונים. נתייחס לנושא המחרוזות, כדי להדגים אחת מהפונקציות:

דוגמה

נתונות שתי תכניות דומות, להדפסת המחרוזת "abc":

```
/* program 1 */

void main()
{
   char *s;

   s="abc";
   printf("%s\n",s);
}

/* program 2 */

void main()
{
   char s[30];
```

```
    strcpy(s,"abc");
    printf("%s\n",s);
}
```

לשתי התכניות פלט זהה, אולם התנהגותן שונה:

בתכנית הראשונה, הפקודה s="abc"; גורמת ל-s להצביע לכתובת המחרוזת "abc" שבטבלת הקבועים, ולא ניתן לשנות את המחרוזת.

בתכנית השנייה המחרוזת "abc" קיימת גם היא בטבלת הקבועים, וניתן להעתיקה מהטבלה למערך התווים בשם s. אולם, מכיוון ש-s אינו מצביע, הפקודה s="abc"; אינה חוקית, ואף לא תעבור כאן הידור.

נקודות לסיכום פרק 7

הנושאים העיקריים שנדונו בפרק זה הם:

- קלט ופלט סטנדרטיים.
- הפונקציה printf.
- רשימת ארגומנטים לפונקציה.
- הפונקציה scanf.
- גישה לקבצים.
- ניהול שגיאות.
- שורות קלט/פלט.

פרק 8. The UNIX System Interface

כדי להבין את המושגים הבסיסיים המוזכרים בהמשך, מומלץ לקרוא את "מדריך לסביבת UNIX" לפני קריאת הפרק הזה. הפרק חשוב במיוחד עבור התלמידים הלומדים בקורס "מעבדה בתכנות מערכות", בשל השימוש בסביבת מערכת ההפעלה דמויית UNIX.

נשים לב כי התכניות המשתמשות באופן ישיר בקריאות מערכת ההפעלה, המתוארות בפרק זה, אינן ניידות (portable). כלומר, לא תמיד ניתן להריצן במערכת הפעלה אחרת, עד אשר יעברו שינויים בהתאמה למערכת. כל זאת בשונה מהשימוש בספרייה הסטנדרטית, שראינו בפרקים קודמים.
קריאות מערכת דומות קיימות בכל מערכת הפעלה, אולם עדיין נותר הצורך בשינוי הקוד, לשם התאמה מדויקת. לעומת חיסרון זה, יתרון השימוש בקריאות המערכת הוא היעילות המרבית של הקוד, בשל הפנייה הישירה למשאבי הגרעין (Kernel) של מערכת ההפעלה.

8.1 File Descriptors

כדי לשנות את מיקום הקלט או הפלט, יש להשתמש ב- < וב- >.
לדוגמה:
בתכנית בשם prog נרצה לקבל את הקלט מקובץ prog_in.txt, ואת הפלט בקובץ prog_out.txt; אזי נבצע:

```
prog < prog_in.txt > prog_out.txt
```

באופן דומה, ניתן להשתמש ב-pipe, כלומר בצינור תהליכי תוכנה, שבו פלט תהליך אחד הוא קלט התהליך הבא (דוגמאות לכך ניתן למצוא ב"מדריך לסביבת UNIX").

8.2 Low Level I/O – Read and Write

נשים לב לשימוש בקובץ ההגדרות "syscalls.h" : הקובץ אינו סטנדרטי, אך הוא מכיל הגדרות נפוצות עבור קריאות מערכת ההפעלה, כגון הגדרת הגודל BUFSIZ.

8.3 Open, Creat, Close, Unlink

תרגיל

יש לבצע את תרגיל 8-1 בספר :

יש לכתוב מחדש את התכנית cat שבפרק 7 (עמוד 162), תוך שימוש בקריאות המערכת (system calls) : read, write, open, close במקום השימוש בפונקציות הספרייה הסטנדרטיות של C.

פתרון (8-1)

```
/*    program cat – concatenate files using UNIX system access */

#include <stdio.h>
#include <fcntl.h>
#define BUFSIZE 1024
void filecopy(int, int);

int main(int argc, char *argv[])
{
  int fd;

  if(argc == 1)
    filecopy(0, 1);
  else {
    while(--argc > 0)
      if(( fd = open(*++argv, O_RDONLY, 0)) == -1) {
            printf("unix cat: can't open %s\n", *argv);
            return 1;
```

```
    }
    else {
            filecopy(fd, 1);
            close(fd);
    }
 }
 return 0;
}
/*  filecopy: copy file "in" to file"out" */
void filecopy(int in, int out)
{
 int n;
 char buf[BUFSIZE];

 while((n=read(in, buf, BUFSIZE)) > 0 )
  write(out, buf, n);
}
```

8.4 Random Access - Lseek

נראה דוגמה לשימוש ב-lseek. שימוש בקריאת מערכת זו מאפשר שיטוט בקובץ, ללא כתיבה או קריאה ממנו. כלומר, הסטת המצביע לנקודה כלשהי בקובץ, ללא כתיבה או קריאה.

הפונקציה הבאה בודקת אם הקובץ תקין, וניתן לבצע עליו חיפוש זה.

```
/* Test if input is capable of seeking */

int seek_test(int fd)
{
   if (lseek(fd, 0, 0) == -1)
      printf("cannot seek\n");
```

```
    else
        printf("seek is OK\n");
    return(0);
}
```

8.5 Example – An implementation of Fopen and Getc

נשים לב לשימוש בקבוע OPEN_MAX, בדוגמה שבעמודים 176-178, להגדרת המספר הגבוה ביותר של קבצים, הפתוחים בו-זמנית. שם ה-ANSI שנקבע עבור קבוע זה הוא FOPEN_MAX ולא OPEN_MAX , כפי שמופיע בספר.

8.6 Example – Listing Directories

נדגיש כי תיקייה (Directory) במערכת UNIX היא בעצם קובץ, ורשימת תכונותיה זהה לתכונותיו של קובץ רגיל: שם, גודל, תאריך יצירה, הרשאות וכו'. תוכן קובץ זה מכיל רשימת קבצים. כך משמשת מערכת ההפעלה גם כמסד נתונים, המאפשר חיפוש מידע על תכונות הקבצים.

8.7 Example – A Storage Allocator

נשים לב שהפונקציות malloc ו-morecore בעמוד 187-188 אינן מוצגות בתכנית אחת. כך הכתוב בהן נכון, אולם ניתן לשפר זאת על ידי כתיבת ההגדרה:

```
static Header *morecore(unsigned);
```

מחוץ לפונקציות (במקום המיקום **בתוך** malloc, המופיע בספר). באופן זה, תחום ההכרה (scope) יהיה רחב יותר, ואילו תחום הקישור (linkage) יהיה ברור יותר.

תרגיל

יש לבצע את תרגיל 6-8 בספר:

פונקציית הספרייה הסטנדרטית calloc(n,size), מחזירה מצביע לשטח זיכרון של מערך של n אובייקטים בגודל size, כאשר תוכן האובייקטים מאותחל ל- 0 (למעשה,

הפונקציה מחזירה מצביע ל-void). יש לכתוב את calloc מחדש, על ידי שימוש בפונקציית הספרייה malloc(size) (אשר מחזירה מצביע לאובייקט בגודל size, כאשר התוכן **לא** מאותחל).

פתרון (6-8)

```
/* The library function calloc(n, size) using malloc */

#include <stdlib.h>
#include <string.h>

void *mycalloc(size_t nobj, size_t size)
{
  void *result = NULL;

  /* use malloc to get the memory */
  result = malloc(nobj * size);

  /* clear the memory */
  if(NULL != result)
    memset(result, 0x00, nobj * size);

  /* return the result */
  return result;
}



/* sample: use of mycalloc */

#include <stdio.h>
#define MSIZE 100
```

```
void main()
{
 int *p = NULL, i = 0;

 p = mycalloc(MSIZE, sizeof *p);

 if(NULL == p)
  printf("mycalloc ERROR.\n");
 else
 {
  for(i = 0; i < MSIZE; i++)
   printf("%X ", p[i]);

  free(p);
 }
}
```

נקודות לסיכום פרק 8

הנושאים העיקריים שנדונו בפרק זה הם:

- שימוש ישיר בקריאות מערכת ההפעלה.
- שינוי המיקום של קלט ופלט סטנדרטיים.
- קריאות המערכת: read, write, open, close.
- קריאת המערכת lseek.
- תיקייה (Directory) במערכת UNIX היא קובץ, ותכונותיה – בהתאם.

נספח א

תכניות לדוגמה

תכנית 1

תכנית זו מקבלת באופן אינטראקטיבי מהמשתמש מספר שלם, ומבצעת חישובים שונים על המספר בעזרת פונקצית עצרת, שמימושה מופיע בסוף התכנית:

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

long int fact (int i);

void main(void)
{

    int i = 0;
    long int j;
    int c;
    printf( "Please enter a number:");
    while(isdigit(c=getchar()))
            i = i*10 + c - '0';
    printf( "Factorial of %d is %ld\n",i,j=fact(i));
    printf( "Double factorial of %d is %ld\n",i,2*j);
    printf( "The value of %ld is %s\n",j,((j%2)?"odd":"even"));

}

/* find the factorial of a number.
  what will happen in function fact if parameter i is negative? */
long int fact (int i)
{
    if (i)
            return (i*fact(i-1));
    else
            return (1);
}
```

דוגמת פלט עבור מספר זוגי:

Please enter a number: 4

Factorial of 4 is 24

Double factorial of 4 is 48

The value of 24 is even

דוגמת פלט עבור מספר אי-זוגי:

Please enter a number: -3

Factorial of 0 is 1

Double factorial of 0 is 2

The value of 1 is odd

תכנית 2

תכנית זו מבצעת פעולה זהה לזו שמבצעת התכנית הקודמת, אלא שכאן המימוש של פונקציית העזר מתומצת יותר.

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

long int fact (int i);

void main(void)
{

    int i = 0;
    long int j;
    int c;
    printf( "Please enter a number:");
    while(isdigit(c=getchar()))
            i = i*10 + c - '0';
    printf( "Factorial of %d is %ld\n",i,j=fact(i));
    printf( "Double factorial of %d is %ld\n",i,2*j);
    printf( "The value of %ld is %s\n",j,((j%2)?"odd":"even"));

}

/* find the factorial of a number.
  what will happen in function fact if parameter i is negative? */
long int fact (int i)
{
    return i ? (i*fact(i-1)) : 1;

}
```

תכנית 3

תכנית זו מקבלת כקלט (מהקלט הסטנדרטי) את התוכן של קובץ C, ומדפיסה את אותה התכנית (לפלט הסטנדרטי) לאחר שההערות הוסרו. התכנית מוחקת הערות חוקיות בשפת C ובשפת C++.

```
#include <stdio.h>

enum status {OUT, IN_STRING, LEFT_SLASH, IN_COMMENT,
RIGHT_STAR, IN_CPP_COM};

void main(void)
{

    int c;
    int state = OUT;
    /* what will happen if we would write instead: enum status state = OUT; ? */

    while((c=getchar()) != EOF)
        switch (state) {
                case OUT:
                  if (c == '/')
                        state = LEFT_SLASH;
                  else {
                        putchar(c);
                        if (c == '\"')
                            state = IN_STRING;
                }
                        break;
               case LEFT_SLASH:
                  if (c == '*')
                        state = IN_COMMENT;
                   else if (c == '/')
```

```
                state = IN_CPP_COM;
                        else {
                putchar('/');
                putchar(c);
                state = OUT;
                }
                break;
        case IN_COMMENT:
                if (c == '*')
                        state = RIGHT_STAR;
                break;
        case IN_CPP_COM:
                if (c == '\n'){
                        state = OUT;
                putchar('\n');
                }
                break;
        case RIGHT_STAR:
                if (c == '/')
                        state = OUT;
                else if (c != '*')
                        state = IN_COMMENT;
                break;
        case IN_STRING:
                if (c == '\"')
                        state = OUT;
                putchar(c);
                break;
        }


}
```

תכנית 4

תכנית זו קולטת באופן אינטראקטיבי מהמשתמש, תו אחר תו. תווים אלו מוכנסים למערך תווי. התכנית הופכת את סדר התווים במערך. בתכנית זו שתי גרסאות למימוש פונקציית ההיפוך (reverse, my_reverse).

```
#include <string.h>
#include <stdio.h>
#define LEN 50

void my_reverse (char s1[], char s2[]);
void reverse (char s[]);

void main(void)
{

    int i;
    int c;
    char x[LEN] = {'\0'};
    char y[LEN] = {'\0'};
    printf( "Please enter a word");
    for (i=0; i<LEN-1 && ((c=getchar()) != '\n'); i++)
            x[i] = c;
    my_reverse(x,y);
    printf( "\nThe reversed version of %s is: %s\n",x,y);
    reverse(y);
    printf( "\nString %s reversed twice is: %s\n",x,y);

}

/* Reverse string str1 to str2. My first version */
void  my_reverse (char str1[], char str2[])
```

```
{

    int i,len;
    len = strlen(str1);
    for (i=0; i<len; i++)
            str2[len-i-1] = str1[i];
    str2[len] = '\0';

}

/* Reverse string s. Second version */
void  reverse (char s[])
{

    int last = strlen(s)-1;
    char temp = s[0];
    if (s[0]) {
            s[0] = s[last];
            s[last] = '\0';
            reverse(s+1);
            s[last] = temp;
    }

}
```

דוגמת ריצה:

```
Please enter a word 12345

The reversed version of 12345 is: 54321

String 12345 reversed twice is: 12345
```

תכנית 5

תכנית זו זהה לתכנית הקודמת, פרט לכך שקריאת תוכן המערך המקורי אינה מבוצעת תו אחר תו, אלא בעזרת פונקציית scanf.

```
#include <string.h>
#include <stdio.h>
#define LEN 50

void my_reverse (char s1[], char s2[]);
void reverse (char s[]);

void main(void)
{

    char x[LEN] = {'\0'};
    char y[LEN] = {'\0'};
    printf( "Please enter a word");
    scanf("%s",x);
    my_reverse(x,y);
    printf( "\nThe reversed version of %s is: %s\n",x,y);
    reverse(y);
    printf( "\nString %s reversed twice is: %s\n",x,y);

}

/* Reverse string str1 to str2. My first version */
void  my_reverse (char str1[], char str2[])
{

    int i,len;
    len = strlen(str1);
    for (i=0; i<len; i++)
            str2[len-i-1] = str1[i];
```

```
    str2[len] = '\0';

}

/* Reverse string s. Second version */
void  reverse (char s[])
{

    int last = strlen(s)-1;
    char temp = s[0];
    if (s[0]) {
            s[0] = s[last];
            s[last] = '\0';
            reverse(s+1);
            s[last] = temp;
    }

}
```

<u>תכנית 6</u>

א. בהינתן שתי מחרוזות DNA שוות אורך, נגדיר **הומולוגיה** בין שתי המחרוזות כאחוז הנוקלאוטידים המתאימים (תווים זהים במקומות התואמים). למשל, עבור שתי המחרוזות האלה:

GTCCAGCTACAT
TTTTCAGATCGA

ההומולוגיה היא: 2/12 = 16.66%.

עליכם לכתוב פונקציה: double homology (char dna_str1[], char dna_str2[])
המחזירה את ההומולוגיה בין שתי מחרוזות אם הן שוות אורך, ומחזירה 1– אם הן בעלות אורך שונה.

ב. נגדיר **הומוביטולוגיה** בין שתי סדרות ביטים כאחוז הביטים המתאימים. למשל עבור:

10110110
11011011

ההומוביטולוגיה היא: 3/8 = 37.5%.

עליכם לכתוב פונקציה: double homobitology (unsigned u1, unsigned u2)
המחזירה את ההומוביטולוגיה בין u1 לבין u2.
הפונקציה צריכה להיות ניידת (portable), ולהצליח לפעול במכונות שונות, שבהן מספר הסיביות בטיפוס unsigned הוא שונה.

ג. עליכם לכתוב פונקציה: unsigned mixed_couple(unsigned n) המקבלת מספר שלם n, ומחזירה מספר שלם, המתקבל ממנו על ידי החלפת מקומותיהן של הסיביות, בתוך כל זוג סיביות, באופן הבא:
אם x הוא: 10110110, אזי mixed_couple יהיה: 01111001.

<u>פתרון:</u>

נשים לב כי הפונקציה שבפתרון מניחה סדר אופרנדים של האופרטור '&' משמאל לימין, דבר שהוא תלוי סביבה ואינו מוגדר ב-ANSI C. <u>תרגיל:</u> יש לכתוב את הפונקציה, באופן מוצלח יותר, ללא הנחה זו.

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

double homobitology(unsigned int u1, unsigned int u2);
int size_int(void);
double homology(char dna_str1[], char dna_str2[]);
unsigned int mixed_couples(unsigned int n);

#define LENGTH 100

void main(void)
{

    char s[LENGTH], t[LENGTH];
    unsigned int a,b;

    printf( "Checking homology function. \nPlease enter a string: ");
    scanf("%s",s);
    printf( "Please enter a second string: ");
    scanf("%s",t);
    printf( "\nThe homology of %s and %s is %f\n",s,t,homology(s,t));

    printf( "Checking homobitology function. \nPlease enter an unsigned
number: ");
    scanf("%u",&a);
    printf( "Please enter a second unsigned number: ");
    scanf("%u",&b);
    printf( "\nThe homobitology of %u and %u is
%f\n",a,b,homobitology(a,b));

    printf( "Checking mixed_couples function. \nPlease enter an unsigned
number: ");
    scanf("%u",&a);
    printf( "\nThe mixed couples of %u is %u. \n",a,mixed_couples(a));
```

```
}

/* homology of strings dna_str1 and dna_str2 */
double homology(char dna_str1[], char dna_str2[])
{

    int i,len;
    double count;

    if ((len = strlen(dna_str1)) != strlen(dna_str2))
            return(-1);
    else
            for (i=0, count=0.0; i<len; i++)
            if (dna_str1[i] == dna_str2[i])
            count++;
    return count / len;

}

/* find size of int */
int size_int(void)
{

    int count,mask = -1;

    for (count=0; mask; mask <<= 1,count++)
            ;

    return count;

}

/* homobitology of numbers: u1 and u2 */
double homobitology(unsigned int u1, unsigned int u2)
```

```
{

    int i,len = size_int();
    double count;

    for (i=0, count=0.0; i<len; ++i, u1>>=1, u2>>=1)
            if ((u1 & 01) == (u2 & 01))
            count++;
    return count / len;

}

/* find mixed couples of int n */
unsigned int mixed_couples(unsigned int n)
{

    unsigned int mask = 01, temp = 0;

    while (mask) {
            temp |= (((mask & n) << 1) | (((mask <<= 1) & n) >> 1));
            mask <<= 1;
    }

    return temp;

}
```

תכנית 7

פתרון נוסף לשאלה הקודמת, כאשר מימוש הפונקציה homobitology שונה.

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

double homobitology(unsigned int u1, unsigned int u2);
int size_int(void);
double homology(char dna_str1[], char dna_str2[]);
unsigned int mixed_couples(unsigned int n);

#define LENGTH 100

void main(void)
{

   char s[LENGTH], t[LENGTH];
   unsigned int a,b;

   printf( "Checking homology function. \nPlease enter a string: ");
   scanf("%s",s);
   printf( "Please enter a second string: ");
   scanf("%s",t);
   printf( "\nThe homology of %s and %s is %f\n",s,t,homology(s,t));

   printf( "Checking homobitology function. \nPlease enter an unsigned
number: ");
   scanf("%u",&a);
   printf( "Please enter a second unsigned number: ");
   scanf("%u",&b);
   printf( "\nThe homobitology of %u and %u is
%f\n",a,b,homobitology(a,b));
```

```
    printf( "Checking mixed_couples function. \nPlease enter an unsigned
number: ");
    scanf("%u",&a);
    printf( "\nThe mixed couples of %u is %u. \n",a,mixed_couples(a));


}

/* homology of strings dna_str1 and dna_str2 */
double homology(char dna_str1[], char dna_str2[])
{

    int i,len;
    double count;

    if ((len = strlen(dna_str1)) != strlen(dna_str2))
            return(-1);
    else
            for (i=0, count=0.0; i<len; i++)
            if (dna_str1[i] == dna_str2[i])
            count++;
    return count / len;

}

/* find size of int */
int size_int(void)
{

    int count,mask = -1;

    for (count=0; mask; mask <<= 1,count++)
            ;
```

```
    return count;

}


/* homobitology of numbers: u1 and u2 */
double homobitology(unsigned int u1, unsigned int u2)
{

    unsigned int temp, len = size_int();
    double count;

    for (temp=(~(u1^u2)),count=0.0; temp; temp>>=1)
            if (temp & 01)
            count++;

    return count / len;

}

/* find mixed couples of int n */
unsigned int mixed_couples(unsigned int n)
{

    unsigned int mask = 01, temp = 0;

    while (mask) {
            temp |= (((mask & n) << 1) | (((mask <<= 1) & n) >> 1));
            mask <<= 1;
    }

    return temp;

}
```

תכנית 8

זהו מימוש דומה למימוש בספר הלימוד (פרק 4) עבור תכנית המפענחת ומבצעת ביטויי postfix. במימוש זה כל התכנית מופיעה בקובץ יחיד.

```
/*
File name: postfix.c
Program: Original postfix program, the whole program in one file.
Contains all function includes, defines and prototytpe declarations
*/

#include <stdio.h>
#include <stdlib.h> /* for atof() */
#include <ctype.h>

#define MAXOP 100  /* max size of operand or operator */
#define MAXVAL 100 /* maximum depth of val stack */
#define NUMBER '0' /* signal that a number was found */

int getop(char []);
void push(double);
double pop(void);
void ungetchh(int);
int getchh(void);

/* reverse Polish calculator */
void main(void)
{
    int type;
    double op2;
    char s[MAXOP];

    while ((type = getop(s)) != EOF) {
    switch (type) {
            case NUMBER:
```

```
                push(atof(s));
                break;
            case '+':
                push(pop() + pop());
                break;
            case '*':
                push(pop() * pop());
                break;
            case '-':
                op2 = pop();
                push(pop() - op2);
                break;
            case '/':
                op2 = pop();
                if (op2)
                    push(pop() / op2);
                else
                    printf("error: zero division.\n");
                break;
            case '\n':
                printf("\t%.4f\n", pop());
                break;
            default:
                printf("error: unknown command %c\n", type);
                break;
            }
    }

}
```

```
int sp = 0; /* next free stack position */
double val[MAXVAL]; /* value stack */
```

```
/* push: push f onto value stack */
void push(double f)
{
    if (sp < MAXVAL)
            val[sp++] = f;
    else
            printf("error: stack full, can't push %.4f.\n", f);
}

/* pop: pop and return top value from stack */
double pop(void)
{
    if (sp > 0)
            return val[--sp];
    else {
            printf("error: stack empty\n");
    return 0.0;
    }
}

/* getop: get next character or numeric operand */
int getop(char s[])
{
    int i=0, c;
    while ((c = getchh()) == ' ' || c == '\t')
    ;
    if (!isdigit(c) && c != '.')
            return c; /* not a number */
    s[i++] = c;
    if (isdigit(c)) /* collect integer part */
            while (isdigit(s[i++] = c = getchh()))
            ;
    if (c == '.') /* collect fraction part */
```

```
            while (isdigit(s[i++] = c = getchh()))
            ;
    s[--i] = '\0';
    if (c != ' ' && c != '\t')
            ungetchh(c);
    return NUMBER;
}


enum {EMPTY, FULL};

int flag = EMPTY, buf;

int getchh(void) /* get a (possibly pushed-back) character */
{
    if (flag == FULL) {
            flag = EMPTY;
            return buf;
    }
    else
            return getchar();
}

void ungetchh(int c) /* push character back on input */
{
    if (flag == EMPTY) {
            buf = c;
            flag = FULL;
    }
    else
            printf( "Buffer is full can not enter more information \n");
}
```

תכנית 9

זוהי הגרסה המחולקת הראשונה לתכנית המפענחת ומבצעת ביטויי postfix.

```
/*
File name: ver1_f1.c
Program: Part of first version of devided postfix program.
Contains main function.
*/

#include "d1.h"

/* reverse Polish calculator */
void main(void)
{
    int type;
    double op2;
    char s[MAXOP];

    while ((type = getop(s)) != EOF) {
    switch (type) {
            case NUMBER:
                push(atof(s));
                break;
            case '+':
                push(pop() + pop());
                break;
            case '*':
                push(pop() * pop());
                break;
            case '-':
                op2 = pop();
                push(pop() - op2);
                break;
            case '/':
```

```
                op2 = pop();
                if (op2)
                        push(pop() / op2);
                else
                        printf("error: zero division.\n");
                break;
        case '\n':
                printf("\t%.4f\n", pop());
                break;
        default:
                printf("error: unknown command %c\n", type);
                break;
        }
    }

}

/*
File name: ver1_f2.c
Program: Part of first version of devided postfix program.
Contains stack functions.
*/

#include "d1.h"



static int sp = 0; /* next free stack position */
static double val[MAXVAL]; /* value stack */

/* push: push f onto value stack */
void push(double f)
{
    if (sp < MAXVAL)
```

```
            val[sp++] = f;
    else
            printf("error: stack full, can't push %.4f.\n", f);
}

/* pop: pop and return top value from stack */
double pop(void)
{
    if (sp > 0)
            return val[--sp];
    else {
            printf("error: stack empty\n");
    return 0.0;
    }
}

/*
File name: ver1_f3.c
Program: Part of first version of devided postfix program.
Contains input functions (getop ext,).
*/

#include "d1.h"


enum {EMPTY, FULL};

int flag = EMPTY, buf;

static int getchh(void) /* get a (possibly pushed-back) character */
{
    if (flag == FULL) {
            flag = EMPTY;
            return buf;
```

```
    }
    else
            return getchar();
}

static void ungetchh(int c) /* push character back on input */
{
    if (flag == EMPTY) {
            buf = c;
            flag = FULL;
    }
    else
            printf( "Buffer is fullc can not enter more information \n");
}

/* getop: get next character or numeric operand */
int getop(char s[])
{
    int i=0, c;
    while ((c = getchh()) == ' ' || c == '\t')
    ;
    if (!isdigit(c) && c != '.')
            return c; /* not a number */
    s[i++] = c;
    if (isdigit(c)) /* collect integer part */
            while (isdigit(s[++i] = c = getchh()))
            ;
    if (c == '.') /* collect fraction part */
            while (isdigit(s[++i] = c = getchh()))
            ;
    s[--i] = '\0';
    if (c != ' ' && c != '\t')
            ungetchh(c);
    return NUMBER;
```

```
}
/*
File name: d1.h
Program: Part of first version of devided postfix program.
Include file: contains includes,defines and prototypes
*/

#include <stdio.h>  /* getchar, printf */
#include <stdlib.h> /* for atof() */
#include <ctype.h>  /* for isdigit() */

#define MAXOP 100  /* max size of operand or operator */
#define MAXVAL 100 /* maximum depth of val stack */
#define NUMBER '0' /* signal that a number was found */

int getop(char []);
void push(double);
double pop(void);
static void ungetchh(int);
static int getchh(void);
```

כדי להריץ את התכנית, ניתן להשתמש בקובץ MAKEFILE המכיל את השורות:

```
all: ver1
ver1: ver1_f1.c ver1_f2.c ver1_f3.c d1.h
gcc ver1_f1.c ver1_f2.c ver1_f3.c -o  ver1
clean: rm  *~
```

תכנית 10

זו הגרסה המחולקת השנייה לתכנית המפענחת ומבצעת ביטויי postfix. לגרסה זו נוספו מספר פעולות. שאלות 3-4, 4-4 בספר הלימוד.

```
/*
File name: ver2_f1.c
Program: Part of second version of devided postfix program.
Contains main function.
The difference between first virsion and second version is:
1. Static variables.
2. Static functions.
3. Register variables.
4. Negative  numbers in input.
5. Operator modulus,   %.
6. Print top of stack.
7. Duplicate top of stack.
8. Switch two top values of stack.
9. Empty stack.
*/

#include "d2.h"

/* reverse Polish calculator */
void main(void)
{
    int type;
    double op1,op2;
    char s[MAXOP];

    while ((type = getop(s)) != EOF) {
    switch (type) {
            case NUMBER:
                push(atof(s));
            break;
```

```
case '+':
    push(pop() + pop());
    break;
case '*':
    push(pop() * pop());
    break;
case '-':
    op2 = pop();
    push(pop() - op2);
    break;
case '/':
    op2 = pop();
    if (op2)
        push(pop() / op2);
    else
        printf("error: zero division.\n");
    break;
case '%':
    op2 = pop();
    if (op2 != 0.0)
        push(fmod(pop(),op2));
    else
        printf("error: zero division.\n");
    break;
case 't':
    printf("\t%.4f\n", top_stack());
    break;
case 'e':
    empty();
    break;
case 'd':
    push(top_stack());
    break;
case 's':
```

```
                op2 = pop();
                op1 = pop();
                push(op2);
                push(op1);
                break;
            case '\n':
                printf("\t%.4f\n", pop());
                break;
            default:
                printf("error: unknown command %c\n", type);
            break;
            }
    }

}


/*
File name: ver2_f2.c
Program: Part of second version of devided postfix program.
Contains stack functions.
*/

#include "d2.h"



static int sp = 0; /* next free stack position */
static double val[MAXVAL]; /* value stack */

/* push: push f onto value stack */
void push(double f)
{
    if (sp < MAXVAL)
```

```
            val[sp++] = f;
    else
            printf("error: stack full, can't push %.4f.\n", f);
}

/* pop: pop and return top value from stack */
double pop(void)
{
    if (sp > 0)
            return val[--sp];
    else {
            printf("error: stack empty\n");
    return 0.0;
    }
}

/* top_stack: print top of stack */
double top_stack(void)
{
    if (sp > 0)
            return val[sp-1];
    else {
            printf("stack is empty, can not print top of stack.\n");
            return 0.0;
    }
}

/* empty: empty the stack */
void empty(void)
{
    sp = 0;
}
```

```
/*
File name: ver2_f3.c
Program: Part of second version of devided postfix program.
Contains input functions (getop ext,).
*/

#include "d2.h"


enum {EMPTY, FULL};

static int flag = EMPTY, buf;

static int getchh(void) /* get a (possibly pushed-back) character */
{
    if (flag == FULL) {
            flag = EMPTY;
            return buf;
    }
    else
            return getchar();
}

static void ungetchh(int c) /* push character back on input */
{
    if (flag == EMPTY) {
            buf = c;
            flag = FULL;
    }
    else
            printf( "Buffer is full can not enter more information \n");
}
```

```
/* getop: get next character or numeric operand */
int getop(char s[])
{
    register int c;

    while ((c = getchh()) == ' ' || c == '\t')
    ;
    if (!isdigit(c) && c != '.' &&  c != '-')
            return c; /* not a number */
    else
    {
            int i = 0;
            if (!isdigit(c = getchh()) && c != '.')
            {
                if (c != ' ' && c == '\t')
                    ungetchh(c);
                return '-';
            }
            else
                s[i++] = c;

            if (isdigit(c)) /* collect integer part */
                while (isdigit(s[i++] = c = getchh()))
                    ;
            if (c == '.') /* collect fraction part */
                while (isdigit(s[i++] = c = getchh()))
                    ;
            s[--i] = '\0';
            if (c != ' ' && c != '\t')
                ungetchh(c);
            return NUMBER;
    }
}
```

```
/*
File name: d2.h
Program: Part of second version of devided postfix program.
Include file: contains includes,defines and prototypes
*/

#include <stdio.h>  /* getchar, printf */
#include <stdlib.h> /* for atof() */
#include <ctype.h>  /* for isdigit() */
#include <math.h>  /* for fmod() */

#define MAXOP 10  /* max size of operand or operator */
#define MAXVAL 10 /* maximum depth of val stack */
#define NUMBER '0' /* signal that a number was found */

int getop(char []);
void push(double);
double pop(void);
double top_stack(void);
void empty(void);
static void ungetchh(int);
static int getchh(void);
```

כדי להריץ את התכנית, ניתן להשתמש בקובץ MAKEFILE המכיל את השורות:

```
all: ver2
ver2: ver2_f1.c ver2_f2.c ver2_f3.c d2.h
gcc ver2_f1.c ver2_f2.c ver2_f3.c  -lm -o ver2
clean: rm *~
```

תכנית 11

תכנית זו מקבלת מהמשתמש, באופן אינטראקטיבי, את התוכן של מערך תווים. תכנית זו מיינת בסדר abc את התווים במערך, בעזרת אלגוריתם המיון "מיון בועות" (bubble sort).

```
/*  Bubble sort */

#include <stdio.h>
#include <string.h>
void bubble_sort(char s[]);
#define LENGTH 100
void main(void)
{
   char str[LENGTH];
   printf("Please enter an unordered string:");
   scanf("%s",str);
   printf("The ordered version of string %s",str);
   bubble_sort(str);
   printf(" is %s \n",str);
}

/* Bubble sort function */
void bubble_sort(char s[])
{
   int n,i,temp;
   for (n=strlen(s)-1;n;--n)
            for(i=0;i<n;i++)
                if (s[i]>s[i+1]){
                    temp = s[i];
                s[i] = s[i+1];
                s[i+1] = temp;
                }

}
```

<u>תכנית 12</u>

זהו מימוש נוסף לאלגוריתם "מיון בועות". החיסרון של המימוש, בדוגמה הקודמת, הוא שניתן למיין בעזרתו רק מערכים תוויים. כאן ניצור קוד כללי יותר. בעזרת מקרו (macro), ניצור "מחולל" לפונקציות, כך שכל אחת מהן מבצעת מיון בועות על מערך המכיל איברים מטיפוס שונה. נשים לב שהאלגוריתם בכל פונקציה זהה. ההבדל הוא בטיפוסי האיברים במערך. בעזרת מקרו נוסף, נגדיר "מחולל" נוסף אשר יכלול פונקציות הדפסה.

```
/* Bubble sort */

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


#define CREATE_BUBBLE(TYPE, FUNC_NAME)\
void FUNC_NAME(TYPE s[], int size)\
{\
    int i;\
    TYPE temp;\
    size--;\
    for (;size;size--)\
            for(i=0;i<size;i++)\
                if (s[i]>s[i+1]){\
                    temp = s[i];\
                    s[i] = s[i+1];\
                    s[i+1] = temp;\
                }\
}

#define CREATE_PRINT_VEC(TYPE, FUNC_NAME, FORMAT)\
void FUNC_NAME(TYPE s[], int size)\
{\
    int i;\
    for(i=0;i<size;i++)\
```

```
    printf(#FORMAT, s[i]);\
    putchar('\n');\
}

CREATE_BUBBLE(int, bubble_int);
CREATE_BUBBLE(char, bubble_char);
CREATE_BUBBLE(float, bubble_float);

CREATE_PRINT_VEC(int,print_int_vec,%d\t);
CREATE_PRINT_VEC(char,print_char_vec,%c\t);
CREATE_PRINT_VEC(float,print_float_vec,%f\t);

void main(void)
{

    char vec1[]="zsdgftebasr";
    int vec2[]={6,4,9,2,4,98,-3,5,7,83};
    float vec3[]={4.5,6.7,3.2,-4.3,5.89};

    printf("The unordered version of vectors:\n\n");
    print_char_vec(vec1,11);
    print_int_vec(vec2,10);
    print_float_vec(vec3,5);

    bubble_int(vec2,10);
    bubble_char(vec1,11);
    bubble_float(vec3,5);

    printf("\n\n\nThe ordered version of vectors:\n\n");
    print_char_vec(vec1,11);
    print_int_vec(vec2,10);
    print_float_vec(vec3,5);

}
```

כך תיראה <u>תכנית 12</u> לאחר פרישת הקריאות ל-macros:

```
/*  Bubble sort */

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

void bubble_int(int s[], int size)
{
    int i;
    int temp;
    size--;
    for (;size;size--)
            for(i=0;i<size;i++)
                if (s[i]>s[i+1]){
                    temp = s[i];
                    s[i] = s[i+1];
                    s[i+1] = temp;
                }
}

void bubble_char(char s[], int size)
{
    int i;
    char temp;
    size--;
    for (;size;size--)
            for(i=0;i<size;i++)
                if (s[i]>s[i+1]){
                    temp = s[i];
                    s[i] = s[i+1];
                    s[i+1] = temp;
                }
}
```

```
void bubble_float(float s[], int size)
{
    int i;
    float temp;
    size--;
    for (;size;size--)
            for(i=0;i<size;i++)
                if (s[i]>s[i+1]){
                    temp = s[i];
                    s[i] = s[i+1];
                    s[i+1] = temp;
                }
}

void print_int_vec(int s[], int size)
{\
    int i;
    for(i=0;i<size;i++)
        printf("%d\t", s[i]);
    putchar('\n');
}

void print_char_vec(char s[], int size)
{\
    int i;
    for(i=0;i<size;i++)
        printf("%c\t", s[i]);
    putchar('\n');
}

void print_float_vec(float s[], int size)
{\
    int i;
    for(i=0;i<size;i++)
```

```
        printf("%f\t", s[i]);
    putchar('\n');
}


void main(void)
{

    char vec1[]="zsdgftebasr";
    int vec2[]={6,4,9,2,4,98,-3,5,7,83};
    float vec3[]={4.5,6.7,3.2,-4.3,5.89};

    printf("The unordered version of vectors:\n\n");
    print_char_vec(vec1,11);
    print_int_vec(vec2,10);
    print_float_vec(vec3,5);

    bubble_int(vec2,10);
    bubble_char(vec1,11);
    bubble_float(vec3,5);

    printf("\n\n\nThe ordered version of vectors:\n\n");
    print_char_vec(vec1,11);
    print_int_vec(vec2,10);
    print_float_vec(vec3,5);

}
```

תכנית 13

זו דוגמה לשימוש במצביעים לצורך העברת פרמטר לפונקציה לפי כתובת (by address). הפונקציה מכניסה לפרמטר הראשון את הערך הגדול מבין ערכי שני הפרמטרים, ולפרמטר השני – את הערך הקטן מבין ערכי שני הפרמטרים.

```
/* Use pointers */
#include <stdio.h>
void max(int *x, int *y);

void main(void)
{
   int a,b;
   printf("Please enter the first integer and then press enter.\n");
   scanf("%d",&a);
   getchar();
   printf("Please enter the second integer and then press enter.\n");
   scanf("%d",&b);
   getchar();
   max(&a,&b);
   printf("Value %d is larger then value %d.\n",a,b);
}

/* Find max*/
void max(int *x, int *y)
{
   int temp;
   if ((*x)<(*y)){
            temp = (*x);
            (*x) = (*y);
            (*y) = temp;
   }

   return;

}
```

תכנית 14

זו דוגמה לשימוש בפונקציית realloc. מקבלים מהמשתמש, באופן אינטראקטיבי, אורך של שרשרת תווית, ואחר כך את תוכנה. תוך שימוש בפונקציית realloc, מקצים בכל פעם כמות זיכרון, בדיוק כנדרש.

```
/*
Use of "realloc"
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LENGTH 10
#define FOREVER for(;;)

void main(void)
{
   int n,i,old_n=0;
   char *begin=(char *)malloc(old_n);
   printf("String begin after initional allocation is at address %p.\n", begin);

   FOREVER {
   fflush(NULL);
   printf("Please enter the length of the string.\n");
   if (!scanf("%d",&n)){
            printf("Input was not an integer, try again\n");
            break;
            }
   if (n>old_n)
            if (!(begin=(char *)realloc(begin, n+1))){
                printf("Not enough memory, string too long, try again\n");
                break;
                }
   fflush(NULL);
```

```
    printf("Please enter the string characters.\n");
    for (i=0;(((*(begin+i)=getchar())!='\n')&&(i<n));i++)
            ;
    *(begin+i+1)='\0';
    printf("\nString begin is %s\nand is at address %p.\n", begin,begin);

    }
    free(begin);
}
```

תכנית 15

זו דוגמה לשימוש ולמעבר על מטריצה, באופן הפשוט ביותר.

```
/*
Use simple matrix
*/

#include <stdio.h>

#define ORIGIN 3
#define DESTINATION 5

void main(void)
{
   float frieght_charges[ORIGIN][DESTINATION];
   int row,col;

   frieght_charges[0][0] = 22.50;
   frieght_charges[0][1] = 43.50;
   frieght_charges[0][2] = 61.10;
   frieght_charges[0][3] = 76.70;
   frieght_charges[0][4] = 89.95;
   frieght_charges[1][0] = 33.50;
   frieght_charges[1][1] = 54.50;
   frieght_charges[1][2] = 56.90;
   frieght_charges[1][3] = 56.25;
   frieght_charges[1][4] = 66.50;
   frieght_charges[2][0] = 49.50;
   frieght_charges[2][1] = 47.50;
   frieght_charges[2][2] = 45.90;
   frieght_charges[2][3] = 44.00;
   frieght_charges[2][4] = 42.20;

   for (row=0;row<ORIGIN;row++) {
```

```
            for (col=0;col<DESTINATION;col++)
            printf("%.2f\t",frieght_charges[row][col]);
            putchar('\n');
    }
}
```

פלט התכנית יהיה:

```
22.50   43.50   61.10   76.70   89.95
33.50   54.50   56.90   56.25   66.50
49.50   47.50   45.90   44.00   42.20
```

תכנית 16

זהו מימוש אחר לדוגמה הקודמת. כאן ניתן לראות דוגמה לשימוש ולמעבר על מטריצה בעזרת מצביעים.

```
/*
Use matrix and pointers
*/

#include <stdio.h>

#define ORIGIN 3
#define DESTINATION 5

void main(void)
{
    int i,j;
    float *p;
    float frieght_charges[ORIGIN][DESTINATION] = {
                {22.50,43.50,61.10,76.70,89.95},
                {33.50,54.50,56.90,56.25,66.50},
                {49.50,47.50,45.90,44.00,42.20}
    };

    for (i=0;i<ORIGIN;i++) {
                for (j=0,p=(float*)(frieght_charges+i);j<DESTINATION;j++)
                    printf("%.2f\t",*(p+j));
                putchar('\n');
    }
}
```

<u>תכנית 17</u>

מימוש אחר לדוגמה הקודמת. גם כאן ניתן לראות דוגמה לשימוש ולמעבר על מטריצה בעזרת מצביעים.

```
/*
Use matrix and pointers
*/

#include <stdio.h>

#define ORIGIN 3
#define DESTINATION 5

void main(void)
{
   int i,j;
   float frieght_charges[ORIGIN+1][DESTINATION+1] = {
            {22.50,43.50,61.10,76.70,89.95,0},
            {33.50,54.50,56.90,56.25,66.50,0},
            {49.50,47.50,45.90,44.00,42.20,0},
            {0   ,0   ,0   ,0   ,0   ,0}
   };

   for (i=0;**(frieght_charges+i);i++) {
            for (j=0;*(*(frieght_charges+i)+j);j++)
                printf("%.2f\t",*(*(frieght_charges+i)+j));
            putchar('\n');
   }
}
```

<u>תכנית 18</u>

זהו מימוש מצביעים. מה יהיה הפלט של התכנית הבאה?

```
/*
Pointers - find output
*/

#include <stdio.h>


void main(void)
{
   char *p[2][3]={"abc","def","gh","ijklmn","opqrstuv","wxyz"};

   putchar(***p);
   putchar((*(*(p+1)+1))[6]);
   putchar(**(p[1]+2));
   /* putchar(*(*(p+1)+1)[6]); this will cause segmentation fault - why? */
   putchar(**p[1]);
   putchar(*(p[1][2]+2));
}
```

תשובה– פלט התכנית יהיה:

auwiy

<u>תכנית 19</u>

זו תכנית המקבלת כארגומנטים של שורת פקודה ייצוג תוי של מספרים בתחום 0 עד 7, ומדפיסה את הייצוג הבינארי המתאים למספר זה.

```
/*
print binary values
*/

#include <stdio.h>
#include <stdlib.h>
void bin_val(int n);

#define MIN 0
#define MAX 7

void main(int argc, char *argv[])
{
   if (argc == 1)
            printf("usage: binval parameter list\n");
   else
            while(--argc)
                bin_val(atoi(*++argv));
}

void bin_val(int n)
{
   static char* array[] =
   {"000","001","010","011","100","101","110","111"};

   if ((n>=MIN) && (n<=MAX))
            printf("The binary value of %d is %s.\n",n,array[n]);
   else
            printf("Error: parameter to function bin_val.\n");

}
```

תכנית 20

מימוש שלישי ואחרון לאלגוריתם "מיון בועות" (bubble sort). מימוש זה הוא גנרי.

```
/*
Generic bubble sort
*/

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

void bubble_sort(void *s, int size, int number,
float (*cmp)(const void *,const void *), void(*swp)(void *,void *,int));

void swp(void *c1, void *c2, int size);

#define LENGTH 100
#define LENGTH_NUM 13
#define LENGTH_REAL 10

#define CMP_BUILD(FUNC_NAME, TYPE)\
float FUNC_NAME(const void *p1,const void *p2)\
{\
   TYPE *s1 = (TYPE *)p1;\
   TYPE *s2 = (TYPE *)p2;\
   return *s1-*s2;\
}

CMP_BUILD(char_cmp, char);
CMP_BUILD(int_cmp, int);
CMP_BUILD(float_cmp, float);


void main(void)
{
```

```
    int i;
    char str[LENGTH]="asjywusgndugo";
    int numbers[LENGTH]={34,67,34,8,2,68,34,7,45,3,67,346,24};
    float real[LENGTH]={5.3,7.44,77.5,66.3,77.3,55.4,8.7,34.7,42.6,3.76};

    printf("The unordered version is: %s\n", str);
    bubble_sort((void *)str,sizeof(char),strlen(str),char_cmp,swp);
    printf("The ordered version is: %s.\n",str);
    putchar('\n');

    printf("The unordered version is:\n");
    for(i=0;i<LENGTH_NUM;i++)
            printf("%d  ", numbers[i]);
    putchar('\n');
    bubble_sort((void *)numbers,sizeof(int),LENGTH_NUM,int_cmp,swp);
    printf("The ordered version is:\n");
    for(i=0;i<LENGTH_NUM;i++)
            printf("%d  ", numbers[i]);
    putchar('\n');

    printf("The unordered version is:\n");
    for(i=0;i<LENGTH_REAL;i++)
            printf("%.2f  ", real[i]);
    putchar('\n');
    bubble_sort((void *)real,sizeof(float),LENGTH_REAL,float_cmp,swp);
    printf("The ordered version is:\n");
    for(i=0;i<LENGTH_REAL;i++)
            printf("%.2f  ", real[i]);
    putchar('\n');

}
```

```
void bubble_sort(void *s, int size, int number,
float (*cmp)(const void *,const void *), void(*swp)(void *,void *,int))
{
    char *array=(char *)s;
    int i;
    for (;number;number--)
            for(i=0;i<number-1;i++)
                if ((*cmp)((void *)(array+i*size),(void *)(array+(i+1)*size))>0)
                (*swp)((void *)(array+i*size),(void *)(array+(i+1)*size),size);
}


void swp(void *p1, void *p2, int size)
{
    char *s1 = (char *)p1;
    char *s2 = (char *)p2;
    char temp;
    int i;

    for(i=0;i<size;i++,s1++,s2++) {
            temp = *s1;
            *s1 = *s2;
            *s2 = temp;
    }

}
```

כך ייראו הפונקציות לאחר פרישת הקריאות ל-macros:

```
float char_cmp(const void *p1,const void *p2)
{
    char *s1 = (char *)p1;
```

```
    char *s2 = (char *)p2;
    return *s1-*s2;
}

float int _cmp(const void *p1,const void *p2)
{
    int *s1 = (int *)p1;
    int *s2 = (int *)p2;
    return *s1-*s2;
}

float float _cmp(const void *p1,const void *p2)
{
    float *s1 = (float *)p1;
    float *s2 = (float *)p2;
    return *s1-*s2;
}
```

תכנית 21

מימוש של עץ בינארי בעזרת מצביע למצביע. מימוש זה יעיל מבחינת זמן ריצה, לעומת המימוש לעץ בינארי, המופיע בספר הלימוד, מכיוון שכאן חוסכים בהצבות מיותרות, בזמן העלייה מהרקורסיה. המימוש מכיל מספרי "קסם", רק לשם הדוגמה. אם נבחר את העץ: 21, 65, 12, 83, 96 , המופיע בתכנית, נראה שצמתים אלו יימחקו. נשים לב כי במערכת UNIX, שבה נשתמש, נבצע "ctr D" עבור EOF.

```
/*
 Binary tree
*/

typedef struct tnode *tree_ptr;
typedef struct tnode {
    int value;
    tree_ptr left;
    tree_ptr right;
    }tree_node;

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

tree_ptr maketree(int val);
void add_tree(tree_ptr* head, int val);
void print_tree(tree_ptr head);
void del_tree(tree_ptr* head, int val);
tree_ptr* find_val(tree_ptr* head, int val);
int minval(tree_ptr head);
void erase_tree(tree_ptr* head);

void main(void)
{
    tree_ptr root=NULL;
    int val, ret_val;
```

```
    while ((ret_val=scanf("%d",&val))!=EOF)
            add_tree(&root,val);

    if (!ret_val){
            printf("Error: input/file\n");
            return;
    }

    printf("\n\nFirst version of tree.\n");
    print_tree(root);

    printf("\n\nSecond version of tree.\n");
    del_tree(&root,96);
    del_tree(&root,83);
    del_tree(&root,12);
    del_tree(&root,65);
    del_tree(&root,21);
    print_tree(root);

    printf("\n\nThird version of tree.\n");
    erase_tree(&root);
    print_tree(root);
    printf("Success\n");
}

void add_tree(tree_ptr* head, int val)
{
    if (!*head)
            (*head) = maketree(val);
    else if ((**head).value < val)
            add_tree(&((**head).right),val);
    else if ((**head).value > val)
            add_tree(&((**head).left),val);
```

```
}

void print_tree(tree_ptr head)
{
    if (head) {
            print_tree(head->left);
            printf("%d\n",head->value);
            print_tree(head->right);
    }
}

tree_ptr maketree(int val)
{
    tree_ptr temp;
    if (!(temp=(tree_ptr)malloc(sizeof(tree_node)))) {
            fprintf(stderr,"Exit: No memory\n");
            exit(1);
    }
    temp->value=val;
    temp->left=temp->right=NULL;
    return temp;
}


void del_tree(tree_ptr* head, int val)
{
    int min;
    tree_ptr* temp=find_val(head, val), t;
    if (!temp) {
            fprintf(stderr,"Value cannot be deleted\n");
            return;
    }
    if (!((**temp).left)) {
            t=(*temp);
```

```
            (*temp)=(**temp).right;
            free(t);
            return;
    }
    if (!((**temp).right)) {
            t=(*temp);
            (*temp)=(**temp).left;
            free(t);
            return;
    }
    min = minval((**temp).right);
    del_tree(&((**temp).right),min);
    (**temp).value = min;

}

tree_ptr* find_val(tree_ptr* head, int val)
{
    if (!*head)
            return NULL;

    if ((**head).value < val)
            return find_val(&((**head).right),val);
    else if ((**head).value > val)
            return find_val(&((**head).left),val);
    else if ((**head).value == val)
            return head;

    return NULL;
}

int minval(tree_ptr head)
{
    while (head->left)
```

```
            head = head->left;

    return head->value;
}

void erase_tree(tree_ptr* head)
{
    if (*head) {
            erase_tree(&((**head).left));
            erase_tree(&((**head).right));
            free(*head);
            (*head) = NULL;
    }
}
```

<u>תכנית 22</u>

זוהי תכנית מחשב אינטראקטיבית הקוראת פקודות, מפענחת ומבצעת אותן.
הפקודות עוסקות בפעולות על מטריצות בגודל 4 על 4. יש להגדיר, תוך שימוש בפקודת typedef, את הטיפוס mat, אשר מסוגל להחזיק מטריצה בגודל 4 על 4. (איברי המטריצה הם מספרים ממשיים).

בנוסף, יש להגדיר 6 משתנים חיצוניים , A, B, C, D, E, F, מטיפוס זה.

כל שם של מטריצה בפקודות הבאות יילקח מתוך השישה הנ"ל.

הפקודות המותרות כקלט לתכנית:

1. **רשימת ערכים ממשיים מופרדים בפסיקים, שם-מטריצה read_mat**

הפקודה תגרום לקריאת הערכים שברשימה לתוך המטריצה ששמה ניתן בפקודה. הערך השמאלי ביותר ברשימה ייקרא לתוך התא (0,0), הערך השני ייקרא לתא (0,1) וכו'. אם ברשימה ישנם פחות מ-16 ערכים, התאים שלא ניתן להם ערך יכילו אפסים. אם ישנם ברשימה יותר מ-16 ערכים, התכנית תתעלם מהערכים העודפים.

לדוגמה: read_mat A,5,6,7 יגרום לתא (0,0) במטריצה A להכיל את הערך 5, לתא (0,1) להכיל את הערך 6, ולתא (0,2) להכיל את הערך 7. יתר תאי מטריצה A יכילו 0.

2. **שם מטריצה print_mat**

המטריצה ששמה ניתן תודפס בצורה המשקפת את מבנה המטריצה.

לדוגמה: הפקודה print_mat A (לאחר ביצוע פקודת ה-read_mat מהסעיף הקודם) תגרום להדפסת הפלט הבא:

	0	1	2	3
0	5	6	7	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

3. **add_mat שם-מטריצה-א', שם-מטריצה-ב'**

מחברת את מטריצה א' ומטריצה ב' ומדפיסה את מטריצת הסכום בפורמט המוגדר בסעיף 2 (print_mat) של דוגמה זו.

4. **add_mat שם-מטריצת-ג', שם-מטריצה-ב' , שם-מטריצה-א'**

מחברת את מטריצה א' ומטריצה ב' ומאכסנת את הסכום במטריצה ג'. פעולה זו אינה גורמת להדפסת פלט.

5. **sub_mat שם-מטריצה-א', שם-מטריצה-ב'**

מחסרת את מטריצה ב' ממטריצה א' ומדפיסה את מטריצת ההפרש בפורמט המוגדר בסעיף 2 (print_mat) של שאלה זו.

6. **sub_mat שם-מטריצת-ג', שם-מטריצה-ב' , שם-מטריצה-א'**

מחסרת את מטריצה ב' ממטריצה א' ומאכסנת את ההפרש במטריצה ג'. פעולה זו אינה גורמת להדפסת פלט.

7. **mul_mat שם-מטריצה-א', שם-מטריצה-ב'**

מכפילה את מטריצה א' ומטריצה ב' ומדפיסה את מטריצת המנה בפורמט המוגדר בסעיף 2 (print_mat) של שאלה זו.

8. **mul_mat שם-מטריצת-ג', שם-מטריצה-ב' , שם-מטריצה-א'**

מכפילה את מטריצה א' ומטריצה ב' ומאכסנת את מטריצת המנה במטריצה ג'. פעולה זו אינה גורמת להדפסת פלט.

9. **mul_scalar ערך-ממשי, שם-מטריצה-א'**
מכפילה את ערכי מטריצה א' בערך הממשי (פרמטר שני) ומדפיסה את מטריצת התוצאה בפורמט המוגדר בסעיף 2 (print_mat) של שאלה זו.

10. **mul_scalar שם-מטריצת ב' , ערך-ממשי , שם-מטריצה-א'**

מכפילה את מטריצה א' בערך הממשי (פרמטר שני) ומאכסנת את התוצאה במטריצה ב'. פעולה זו אינה גורמת להדפסת פלט.

11. **trans_mat שם-מטריצה-א'**

מבצע "היפוך" (transpose) על ערכי מטריצה א', ומדפיסה את מטריצת התוצאה בפורמט המוגדר בסעיף 2 (print_mat) של שאלה זו. לא חל שינוי בערכי מטריצה א' עצמם.

12. **trans_mat שם מטריצה ב' , שם-מטריצה-א'**

מבצע "היפוך" (transpose) על ערכי מטריצה א', ומאכסן את המטריצה "ההפוכה" במטריצה ב'. פעולה זו אינה גורמת להדפסת פלט. לא חל שינוי בערכי מטריצה א' עצמם.

13. **stop**

התכנית תפסיק לרוץ ותצא למערכת ההפעלה.

התכנית צריכה לתת סימן (prompt) על המסך, המודיע על מוכנות לקבלת קלט.
התכנית תמשיך לעבוד עד שתקבל את הפקודה stop.
התכנית אינה מניחה נכונות הקלט, ויש להודיע על שגיאת קלט.

דוגמאות

לפקודה:

read_mat W, 3.2, 8

יש להגיב בהודעה:

"No such matrix name"

לפקודה:

kkkk A, B, C

יש להגיב בהודעה:

“No such command"

לפקודה:

read_mat A, B,567

יש להגיב:

“Wrong parameters, second parameter must be a real number”

וכו'...

<u>פתרון</u>: מימוש חלקי לתכנית 22

```
/*
 Matrix commands
*/

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

#define SIZE 4
#define FOREVER for(;;)
#define OK 1
enum {NO,YES};

typedef int matrix[SIZE][SIZE];

matrix a,b,c,d,e,f;

struct {
    char name;
    matrix *mat;
    } mats[]={
                {'a',&a},
                {'b',&b},
```

```
            {'c',&c},
            {'d',&d},
            {'e',&e},
            {'f',&f},
            {'A',&a},
            {'B',&b},
            {'C',&c},
            {'D',&d},
            {'E',&e},
            {'F',&f},
            {'#',NULL}
    };

typedef struct tnode {
    matrix *first_parm;
    matrix *second_parm;
    matrix *result;
    }parms;

parms temp_parm;

void read_mat(void);
void print_mat(void);
void add_mat(void);
void sub_mat(void);
void mul_mat(void);
void trans_mat(void);
void stop(void);

matrix *get_parameters(void);
void get_data(matrix *mat);
void prn_mat(matrix *mat);

struct {
```

```
    char *name;
    void (*func)(void);
    } cmd[]={
            {"read_mat",read_mat},
            {"print_mat",print_mat},
            {"add_mat",add_mat},
            {"sub_mat",sub_mat},
            {"mul_mat",mul_mat},
            {"trans_mat",trans_mat},
            {"stop",stop},
            {"not_valid",NULL}
    };


void main(void)
{
    char command[30];
    int i;
    FOREVER
            if (scanf("%s",command)==1) {
                for(i=0; cmd[i].func!=NULL; i++)
                    if (strcmp(command,cmd[i].name)==0)
                        break;

                if (cmd[i].func==NULL)
                    fprintf(stderr,"Command does not exit:%s\n",command);
                else
                    (*(cmd[i].func))();

            }
}
```

תכנית 23

זהו מימוש של union. מה יהיה פלט התכנית?

```
/*
 union
*/

#include <stdio.h>
void main(void)
{

    union {
            int index;
            struct{
            unsigned int x:2;
            unsigned int y:1;
            unsigned int z:1;
            } bits;
    } number;

    for (number.index=3;number.index<8;number.index++)
            printf("%d\t%d\t%d\t%d\n",
    number.index,number.bits.x,number.bits.y,number.bits.z);

}
```

<u>פתרון</u>

באופן עקרוני, התשובה תלוית מחשב: אם פריסת שדות הסיביות היא משמאל לימין, אין התנגשות בין הערכים שהוכנסו למשתנה index לבין שדות הסיביות x,y,z . אולם אם הפריסה של שדות הסיביות היא מימין לשמאל (כמו ב-UNIX), נקבל את הפלט הזה:

3	3	0	0
4	0	1	0
5	1	1	0
6	2	1	0
7	3	1	0

<u>תכנית 24</u>

תכנית זו משלבת רשומות, structures, איחודים, unions, ושדות סיביות, bit fields. התכנית מחשבת את המשכורות של עובדים שונים במפעל. סוגי העובדים הם: עובד זמני, עובד רגיל, איש מכירות. משכורות העובדים מחושבות על פי סוגים אלו, באופן הבא:

עובד זמני – חישוב על פי מספר שעות העבודה החודשיות.

עובד רגיל – משכורת בסיסית שעליה נוספים 100 שקלים לעובד נשוי, 200 שקלים לעובד גרוש, 300 שקלים לעובד אלמן, וכן 100 שקלים עבור כל ילד מילדיו של העובד.

איש מכירות – גובה העמלה * היקף המכירות החודשי שלו

יש לשים לב: התכנית משתמשת ב-goto. יש לזכור שהשימוש בכך אינו מומלץ, ומופיע כאן לשם הדוגמה בלבד.

<u>תרגיל רשות</u>: שינוי התכנית כך שלא תכיל goto

```
/*
 unions bit fields and structures
*/

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

enum {TEMP,REG,SALES};
enum {SINGLE,MARRIED,DIVORCED,WIDOWED};

typedef struct {
    char first_name[30];
    char last_name[30];
    long int id_number;
} person;

typedef struct {
    int rate;
    int num_hours;
} temp_emp;
```

```
typedef struct {
    int salary;
    struct{
            unsigned int married:2;
            unsigned int children:4;
    } status;
} reg_emp;

typedef struct {
    float comission;
    long int sales;
} sales_emp;

typedef struct {
    int type;
    person p;
    union{
            temp_emp t;
            reg_emp r;
            sales_emp s;
    } emp;
} employee;

#define NUM_EMP 3


char general_data(int i, employee table[]);
void temp_record(int i, employee table[]);
void reg_record(int i, employee table[]);
void sales_record(int i, employee table[]);
void salaries(employee table[]);
void clrscr(void);
```

```
void main(void)
{
    int i;
    employee table[NUM_EMP];
    char temp;
    for (i=0;i<NUM_EMP;i++) {
            temp = general_data(i,table);
            switch (temp) {
            case 'T': temp_record(i,table);
                  break;
            case 'R': reg_record(i,table);
                  break;
            case 'S': sales_record(i,table);
                  break;
            }
    }
    salaries(table);
}

char general_data(int i, employee table[])
{
    char temp[2];
    clrscr();
    printf("Please enter last name of employee, and then return.\n");
    if (scanf("%s",table[i].p.last_name)!=1)
            goto error;
    printf("Please enter first name of employee, and then return.\n");
    if (scanf("%s",table[i].p.first_name)!=1)
            goto error;
    printf("Please enter id of employee, and then return.\n");
    if (scanf("%ld",&table[i].p.id_number)!=1)
            goto error;
    printf("Please enter type of employee:\n");
    printf("\nT for temporary employee\n");
```

```
    printf("R for regular employee\n");
    printf("S for sales person\n");
    printf("and then press enter.\n");
    if (scanf("%s",temp)!=1)
            goto error;

    return *temp;
    error: printf("ERROR: bad input, program teerminated.\n");
    exit(1);

}

void temp_record(int i, employee table[])
{
    clrscr();
    table[i].type = TEMP;
    printf("Please enter rate per hour, and then return.\n");
    scanf("%d",&table[i].emp.t.rate);

    printf("Please enter number of hours this month,\n");
    printf("and then press enter.\n");
    scanf("%d",&table[i].emp.t.num_hours);
}

void reg_record(int i, employee table[])
{
    char temp[10];
    int j;
    clrscr();
    table[i].type = REG;
    printf("Please enter salary of employee, and then return.\n");
    scanf("%d",&table[i].emp.r.salary);
```

```
    printf("\nPlease enter marriage status:\n");
    printf("\nS for single\nM for married\n");
    printf("D for divorced\nW for widowed\n");
    printf("Please press enter.\n");
    scanf("%s",temp);
    switch(*temp) {
            case 'S': table[i].emp.r.status.married=SINGLE;
                  break;
            case 'M': table[i].emp.r.status.married=MARRIED;
                  break;
            case 'D': table[i].emp.r.status.married=DIVORCED;
                  break;
            case 'W': table[i].emp.r.status.married=WIDOWED;
                  break;

    }

    printf("Please enter number of children (0 for no children).\n");
    printf("Please press enter.\n");
    scanf("%d",&j);
    table[i].emp.r.status.children=j;

}

void sales_record(int i, employee table[])
{
    clrscr();
    table[i].type = SALES;
    printf("Please enter the comission, and then return.\n");
    scanf("%f",&table[i].emp.s.comission);

    printf("Please enter sum of sales this month,\n");
    printf("and then press enter.\n");
    scanf("%ld",&table[i].emp.s.sales);
```

```
}

void salaries(employee table[])
{

    int i,j;
    clrscr();
    for (i=0;i<NUM_EMP;i++) {
            printf("\nEmployee number: %ld\n",table[i].p.id_number);
    switch(table[i].type) {
                case TEMP:
                    printf("\nSalary per hour: %d\n",table[i].emp.t.rate);
                    printf("\nNumber of hours this month: %d\n",table[i].emp.t.num_hours);
                    printf("\nSalary is: %d\n",table[i].emp.t.rate * table[i].emp.t.num_hours);
                break;
                case REG:
                    printf("\nBase salary is: %d\n",table[i].emp.r.salary);
                    printf("Employee is ");
                    switch(table[i].emp.r.status.married) {
                        case SINGLE: printf("single.\n");
                                     j=0;
                                     break;
                        case MARRIED: printf("married.\n");
                                      j=100;
                                      break;
                        case DIVORCED: printf("divorced.\n");
                                       j=200;
                                       break;
                        case WIDOWED: printf("widowed.\n");
                                       j=300;
                                       break;
                    }
```

```
                printf("The employee has %d children.\n", table[i].emp.r.status.children);
                printf("The salary is %d.\n",
                table[i].emp.r.salary+j+table[i].emp.r.status.children*100);
                break;
            case SALES:
                printf("The comission is: %f\n",table[i].emp.s.comission);
                printf("Amount of sales this month: %ld\n",table[i].emp.s.sales);
                printf("Salary is: %f.\n",
                table[i].emp.s.comission/100*table[i].emp.s.sales);
                break;

            }
    }
}

void clrscr(void)
{
    system("clear");
}
```

<u>תכנית 25</u>

תכנית זו מקבלת מהמשתמש, באופן אינטראקטיבי, מספר שלם אי-זוגי, ומציירת צורה הדומה לשעון חול.

נראה כאן ארבעה מימושים אפשריים לפונקציה print_pir, המבצעת את הציור.

דוגמה לציור המתקבל עבור n=9 :

```
*********
 *******
  *****
   ***
    *
   ***
  *****
 *******
*********
```

```
/*
 Print with '*'
*/

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

#define MAXNUM 50

void print_pir(int);

void main(void)
{
    char number[MAXNUM];
    int num,i;
    printf("Please enter an odd number, and then press return.\n");
```

```
    for (i=0;i<MAXNUM && ((number[i]=getchar())!='\n');i++)
            ;
    if (((num=atoi(number))!=0) && ((num%2)==1))
            print_pir(num);
    else
            printf("Eroor: bad input, return.\n");
}

/* version 1 */
void print_pir(int n)
{
    int i,k,lim;
    for (i=0,lim=(n+1)/2;i<lim;i++) {
            for (k=0;k<i;k++)
                putchar(' ');
            for (k=0;k<n-2*i;k++)
                putchar('*');
            putchar('\n');
    }

    for (lim=(n-1)/2-1;lim>=0;lim--) {
            for (k=0;k<lim;k++)
                putchar(' ');
            for (k=0;k<n-lim*2;k++)
                putchar('*');
            putchar('\n');
    }
}

/* version 2 */
/*void print_pir(int n)
{
    int i,j,k,lim;
```

```
    for (i=0,j=n-1;j>=0;i++,j--) {
            for (k=0,lim=(i<j)?i:j;k<lim;k++)
                putchar(' ');
            for (k=0;k<abs(j-i)+1;k++)
                putchar('*');
            putchar('\n');
    }

}
*/

/* version 3: good until 31 only */
/*void print_pir(int n)
{
    int i,j;
    #define STARS "*****************************************"
    #define BLANKS "                "
    for (i=0,j=n-1;j>=0;i++,j--)
            printf("%.*s%.*s\n",i<j?i:j,BLANKS,abs(i-j)+1,STARS);

}
*/

/* version 4 : good until 41 only */
/*void print_pir(int n)
{
    int i,j;
    #define STARS "*****************************************"
    for (i=0,j=n-1;j>=0;i++,j--)
            printf("%*.*s\n",i<j?j+1:i+1,abs(i-j)+1,STARS);

}
*/
```

תכנית 26

תכנית זו מקבלת כארגומנטים של שורת פקודה רשימה של שמות קבצים.

עבור כל קובץ קלט (ששמו מופיע ברשימת הארגומנטים של שורת הפקודה), היא יוצרת קובץ פלט, בעל שם זהה, פרט לסיומת (סיומת p.). ההנחה היא שקובצי הקלט (הקבצים המופיעים ברשימת הארגומנטים של שורת פקודה) מכילים תאריכים, באחד משלשת הפורמטים הבאים:

dd.mm.yy
dd.mm.yyyy
dd/mm/yy
dd/mm/yyyy
dd month yy
dd month yyyy

יש לשים לב שייתכן שיש בקובצי הקלט קלט שגוי. התכנית מניחה שכל תאריך מופיע בשורה נפרדת בקבצים. התכנית עוברת על קובצי הקלט, ועבור כל קובץ קלט – מדפיסה לקובץ הפלט המתאים את התאריכים שנקראו בפורמט dd.mm.yyyy, ללא קשר לפורמט שבו הופיע התאריך בקובץ הקלט.

```
/*
 Build new file name
*/

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

#define OK 0
#define MAX_LINE 100
#define MONTH_LENGTH 15

void change_date(FILE *, FILE *);
void build_new_file_name(char *string);
int legal_date(int day, int month, int year);
```

```
int main(int argc, char *argv[])
{
   FILE * ifp, *ofp;

   if (argc == 1)
            return !OK;

   while (--argc > 0)
            if ((ifp=fopen(*++argv,"r")) == NULL)
              printf("Cannot open file %s, continue with next
file.\n\n",*argv);
            else {
                  build_new_file_name(*argv);
                  if ((ofp=fopen(*argv,"w"))==NULL) {
                     printf("Cannot open file %s, continue with next file.\n\n",*argv);
                     fclose(ifp);
            }
             else {
                 change_date(ifp,ofp);
                 fclose(ifp);
                 fclose(ofp);
             }
            }

   return OK;
}

void change_date(FILE *ifp, FILE *ofp)
{
   char line[MAX_LINE], name_month[MONTH_LENGTH],*p;
   int day, month, year;
   char *tab_names[]= {
            "",
            "jan",
```

```
        "feb",
        "mar",
        "apr",
        "may",
        "jun",
        "jul",
        "aug",
        "sep",
        "oct",
        "nov",
        "dec",
        "illegal month",
};
int num_months=sizeof(tab_names)/sizeof(char *);

while (fgets(line,MAX_LINE,ifp)!=NULL) {
        *(strchr(line,'\n'))='\0';
        if (sscanf(line,"%d.%d.%d",&day,&month,&year)==3);
            else if (sscanf(line,"%d/%d/%d",&day,&month,&year)==3);
                else if (sscanf(line,"%d%s%d",&day,name_month,&year)==3) {
            p=name_month;
            while (*p)
                *p++=tolower(*p);
            for(month=1; month<num_months;month++)
                if (!strncmp(name_month,tab_names[month],3))
                    break;
            if (month>12) {
                fprintf(stderr,"Illegal month %s, cannot translate date.\n\n",name_month);
                continue;
            }
        }
        else {
            fprintf(stderr,"Illegal format %s, cannot translate date.\n\n",line);
            continue;
```

```
            }
            year %=100;
            year += (year <50) ? 2000 : 1900;
            if (legal_date(day,month,year) == OK)
               fprintf(ofp,"%d.%d.%d\n\n",day,month,year);
   }

}

void build_new_file_name(char *string)
{
   char *cp;

   cp = strchr(string,'.');

   *++cp = 'p';

   (*++cp)='\0';

}

int legal_date(int day, int month, int year)
{
   static char daytab[2][13] = {
            {0,31,28,31,30,31,30,31,31,30,31,30,31},
            {0,31,29,31,30,31,30,31,31,30,31,30,31}
   };

   int leap;

   if (month<1 || month>12) {
     fprintf(stderr,"%d.%d.%d,Illegal month, not printed in output
   file.\n\n",day,month,year);
     return !OK;
```

```
    }

    if (year < 0) {
      fprintf(stderr,"%d.%d.%d,Illegal year not printed in output
    file.\n\n",day,month,year);
      return !OK;
    }

    leap=year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
    if ((0 > day) || (day > daytab[leap][month])) {
      fprintf(stderr,"%d.%d.%d,Illegal day in month, for given month and
    year,\n\t not printed in output.\n",day,month,year);
      return !OK;
    }

    return OK;
}
```

<u>תכנית 27</u>

תכנית זו מראה הגדרות של מספר macros:

FOREVER – לולאה אינסופית

ABSOLUTE – ערך מוחלט

ISDIGIT – האם תו נתון הוא תו של ספרה

SQUARE – העלאת מספר בריבוע

בהמשך מופיע מימוש של פעולת החלפה (swap), תחילה כפונקציה ואחר כך כ-macro. לבסוף מופיעה פונקציית main, המכילה זימונים לשני המימושים השונים של פעולת ההחלפה. (<u>תרגיל</u>: יש לרשום מהם היתרונות והחסרונות של כל אחד מהמימושים.)

```
/*
 macros
*/

#include <stdio.h>

#define FOREVER for(;;)
#define ABSOLUTE(A) (A)>0?(A):-(A)
#define ISDIGIT(C) (((C)>='0')&&((C)<='9'))?1:0
#define SQUARE(X) (X)*(X)

/* version 1 : with bug */
void bad_swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}

/* version 2 : OK */
void swap(int *x, int *y)
{
    int temp = *x;
```

```
    *x = *y;
    *y = temp;
}

/* version 3 : macro  */
#define SWAP(T,X,Y)\
{\
    T temp;\
    temp = X;\
    X = Y;\
    Y = temp;\
}


void main(void)
{
    int a=1,b=2;

    printf("1. before swap: a=%d b=%d\n",a,b);
    swap(&a,&b);
    printf("2. after swap: a=%d b=%d\n",a,b);
    SWAP(float,a,b);
    printf("3. after another swap: a=%d b=%d\n",a,b);
    bad_swap(a,b);
    printf("2. after bad swap: a=%d b=%d\n",a,b);

}
```

תוצאות הרצת התכנית יהיו:

```
1. before swap: a=1 b=2
2. after swap: a=2 b=1
3. after another swap: a=1 b=2
4. after bad swap: a=1 b=2
```

נספח ב

תרגילים ודוגמאות

<u>תרגיל 1</u>

התכנית הבאה מכילה שגיאה. יש למצוא אותה. מה יהיה פלט התכנית ומדוע?
ניתן להניח שהקלט המוקלד הוא המספר 13.

```
/*
 input / output
*/

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


void main(void)
{
    int i,j;
    unsigned char *p, temp;

    if (!scanf("%d",&i)) {
            fprintf(stderr,"Wrong input\n");
            exit(1);
    }
    p = (unsigned char *)calloc(i,sizeof(unsigned char));
    for (j = i;j;j--)
            *(p+j) = j;
    for (j = i;j;j--)
            printf("%d\t",(int)(*(p+j)));
    putchar('\n');
    for (j = i;j;j--)
            *(p+i) = (temp = p[i] << 2) && (temp >> 2);
    for ( ;j;j--)
    printf("%d\t", (int)(*(p+j)));
}
```

<u>פתרון</u>

השגיאה: בתחילת הלולאות j=13 , כך ניגשים למקום (p+j)*, כלומר הלולאה משתמשת באינדקס 13 במערך, כאשר מוגדרים רק 13 מקומות (בין 0 ל-12).

מודפסים מספרים בסדר יורד. הפלט יהיה:

13 12 11 10 9 8 7 6 5 4 3 2 1

תרגיל 2

מה יהיה הפלט של התכנית הזאת?

```
/*
 output
*/

#include <stdio.h>

int func1(int a);
float func2(int a);
void func3(int a);


void main(void)
{
    int x=5, y=223;
    func1(x);
    func2(x);
    func3(x);
    func1(y);
    func2(y);
    func3(y);
}

int func1(int a)
{
    auto float b;
    b = (float)a;
    printf("%f\n",b);
    return 0;
}

float func2(int a)
```

```
{
    float b;
    b = a;
    printf("%f\n",b);
    return 0.0;
}

void func3(int a)
{
    register b;
    b = (double)a;
    printf("%f\n",(float)b);
}
```

<u>פתרון</u>

מודפסים מספרים. הפלט יהיה:

```
5.000000
5.000000
5.000000
223.000000
223.000000
223.000000
```

<u>שאלה</u>

האם יש עדיפות לאחת הפונקציות לשם מימוש התוצאה (יתרון בזמן ריצה / זיכרון)?

תרגיל 3

מה יהיה הפלט של התכנית הזאת?

```
/*
 output
*/

#include <stdio.h>

void main(void)
{
    char vec[] = "abcdefghij";
    putchar(vec[vec[1] -vec[0]] +=1);
}
```

פתרון

הפלט יהיה התו c. הסבירו מדוע!

<u>תרגיל 4</u>

מה יהיה הפלט של התכנית הזאת?

```
/*
 output
*/

#include <stdio.h>

void main(void)
{
    int a,b;
    for (a=1, b=0; a; a<<=1, b++);
    printf("%d\n",b);
}
```

<u>פתרון</u>

הפלט יהיה המספר 32. הסבירו מדוע!

<u>תרגיל 5</u>

נתונה התכנית הבאה, המחולקת למספר קבצים (main.c, file1.c, file2.c, head.h). יש למצוא את כל השגיאות בתכנית.

הקבצים הם:

1. קובץ main.c :

```
/*
 find what is worng.
main.c file
*/

#include "head.h"
extern int g;

void main(void)
{
    int a='a',b='b',c='c';
    char d='d',e='e',f='f';

    c=f1(a);
    a=g+f2(b,b);
    b=f3(a,b,c);
    f4(d);
    b=f5(a);
}
```

2. קובץ file1.c :

```
/*
 file1.c
*/

#include "head.h"
extern int g;
extern int h=55;

void f1(int a)
{
    int b;
    if (scanf("%d",b)==1)
                fprintf("%d\n",a*b);
    return;
}

f2(int a, int b)
{
    a = g * 5;
    b = b * a;
}
```

3. קובץ file2.c :

```
/*
 file2.c
*/

extern int h=33;

int f3(char a, char b, char c)
{
    int k,l;
    k = a + g;
```

```
    l = b + h;
    return k+l+c;
}

f4(char a)
{
    return a+1;
}
```

4. קובץ head.h:

```
/*
 head.h
*/

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

void f1(int);
int f2(int, int);
int f3(char, char, char);
int f4(char);

int g=44;

int f5(int a)
{
    return 2*a;
}
```

תרגיל 6

נתונה התכנית הבאה. הסבירו במילים מה עושה התכנית, ובפרט מה עושה הפונקציה "func", **וציינו מה הפלט שלה. ניתן להניח כי הרשימה הנבנית מכילה את המספרים:** 5,1,32,5,5,5,33,5,25,5,0 .

```
typedef struct k {
    int value;
    struct k *next;
    struct k *prev;
    }node;

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

#define SIZE 30

int func(node *);
node *build_list(FILE *);
void print_list(node *);
FILE *my_open(char *name, char *mode);


void main(int argc, char *argv[])
{
    node *h=NULL;
    int c;
    FILE *file1;


    if (argc !=2)
    {
            printf("Usage: %s file\n", argv[0]);
            exit(2);
    }
```

```
    /* read file and build list */
    file1 = my_open(argv[1], "r");

    h=build_list(file1);
    printf("before func:\n");
    print_list(h);
    c=func(h);
    printf("after func:\n");
    print_list(h);

    printf("\n%d\n",c);
}

int func(node *head)
{
    node *p, *q, *temp;
    int count;
    for (p=head, count=0; p; p=p->next, count++)
            for (q=p->next; q; q=q->next)
                if (p->value == q->value) {
                    q->prev->next = q->next;
                    if (q->next)
                        q->next->prev = q->prev;
                    temp = q->prev;
                    free(q);
                    q = temp;
                }
    return count;
}

void print_list(node *h)
{
    for (;h; h=h->next)
            printf("%d\t",h->value);
```

```
    }

/* build list from file with: 5,1,32,5,5,5,33,5,25,5,0  */
node * build_list(FILE *file1)
{

    node *p, *head, *prev;
    int infile1=1;

    /* read file to build list */

    head = prev = p = malloc(sizeof(node));
    while ((p != NULL) && (infile1 != EOF))
    {
            infile1 = fscanf( file1, "%d", &(p->value));

            if (infile1 != EOF)
            {
                    /* Output data read: */
                    printf( "%d\n", p->value );

                    prev = p;
                    p->next = malloc(sizeof(node));
                    p = p->next;
                    if (p != NULL)
                        p->prev = prev;
            }
            else p->next = NULL;
    }

    fclose(file1);

    return head;
```

```
}

FILE *my_open(char *name, char *mode)
{
   FILE *fp;

   char *msg[SIZE];
   fp = fopen(name,mode);
   if (fp == NULL) {
            printf("can not open %s", name);
            exit(1);
   }
   return(fp);
}
```

<u>פתרון</u>

הפלט יהיה:

```
before func:
0  5  25  5  33  5  5  5  32  1  5
after func:
0  25  33  32  1  5
6
```

התכנית בונה רשימה מקובץ, כאשר כל מספר בקובץ מופיע בשורה חדשה. בעזרת הפונקציה "func " נמחקים המספרים החוזרים ברשימה. מודפסת הרשימה לפני השינוי ואחריו. לבסוף מודפס מספר האיברים ברשימה החדשה.

תרגיל 7

מה יהיה הפלט של התכנית הזאת?

```
/*
 output
*/

#include <stdio.h>

int r;
float f;

void main(void)
{

    void *p;
    p = (void *)&r;
    *(int *)p = 2;

    printf("%d\n",*((int *)p));

    p = (void *)&f;
    *(float *)p = 1.1;

    printf("%f\n",*((float *)p));

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

```
2
1.100000
```

תרגיל 8

נניח כי במחשב שלנו כתובת מיוצגת בארבעה בתים.

בתכנית הבאה מופיעות ההגדרות, ולאחריהן מודפסים ביטויים. מה יהיה הפלט המודפס?

```
/*
 sizeof
*/

#include <stdio.h>

char a, *b=&a, c[]="abcdefg", *d[] = {"abc","def","ghij","k",NULL};

void main(void)
{


   printf("1. size of: a=%d b=%d c=%d
d=%d\n",sizeof(a),sizeof(b),sizeof(c),sizeof(d));
   printf("2. size of: *b=%d *c=%d *d=%d\n",sizeof(*b),sizeof(*c),sizeof(*d));
   printf("3. *(d+4)=%d\n",*(d+4));

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

1. size of: a=1 b=4 c=8 d=20
2. size of: *b=1 *c=1 *d=4
3. *(d+4)=0

תרגיל 9

מה עושה הפונקציה "what" **בתכנית הזאת?**

```
/*
 what
*/

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

int what(char *a, char *b)
{
    int i,j,x,f=0;
    for (j=0; j<=strlen(a)-strlen(b); ++j) {
            for (x=0, i=j; i<=j+strlen(b)-1; ++i)
                if (b[i-j] == a[i])
                    ++x;
            if (x == strlen(b))
                    ++f;
    }
    return (f>0)?1:0;
}

void main(void)
{

    char str1[]="123456789";
    char str2[]="12345";

    printf("\n%d\n",what(str1, str2));

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

1

תרגיל 10

התכנית הבאה מכילה שגיאות. יש למצוא אותן.

```
/*
 Student grade reporting program
Input: four fields from a file:name, student id, two grades calculations:
compute weighted average for all students, identify data input errors with
a macro, loop for all
Output: student grade report, name, student id, average grade, overall
average, data exception messages.
*/

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

#defineVALID_GRADES((x),(y)) ((x)>=0)&&((x)<=100)&&((y)>=0)&&((y)<=100)
#define LENGTH 30

void main(void)
{
   FILE *fp;
   char onechar, *in_fname;
   char name[LENGTH];
   char stu_id[LENGTH];
   int grade1, grade2, no_stu=0;
   float average, oa_avg=0.00;

   puts("\t\tStudent Grade Report");
   putchar('\a');  /* sounds alarm */

   do
   {
            puts("Are you ready to continue -- ,<Y>,<N>,<Q>uit?");
            onechar = toupper(getchar());
            fflush(stdin);
```

```
    }

    while (onechar != 'Q' && onechar != 'Y');
        if (onechar == 'Q')
            exit(1);
    puts("Enter the input filename to process");
    puts("including the path if necessary");
    puts("followed by the <Enter> key");
    gets(in_fname);
    fp = fopen(in_fname,"r");
    if (fp == NULL) {
            puts("The input file is not available. Possible errors are:");
            puts("a bad file name, a bad path, or an open disk drive.");
            exit(2);
    }

    printf("\n NAME        STUDENT'S ID     GRADE");
    while (fscanf(in_fname,"%s %s %d %d",name, stu_id, grade1, grade2) !=
EOF) {
            if (VALIDE_GRADES(grade1, grade2)) {
                no_stu++;
                average = (0.4 * grade1) + (0.6 * grade2);
                oa_avg+=average;
                printf("\n *%-20s%11s %9.1f", name, stu_id, average);
            }
            else printf("\n *%-20s%11s CHECK GRADES %d %d",
                            name, stu_id, grade1, grade2);
    }
    printf("\n\n NO.STUDENTS %d CLASS AVERAGE %9.1f \n\n",
                    no_stu, oa_avg/no_stu);

    fclose(fp);
    rerturn(0);
}
```

תרגיל 11

מה עושה הפונקציה "what" **בתכנית הזאת?**

```
/*
 return output
*/

#include <stdio.h>

int what(char str[], char c)
{
    char *ptr;

    for (ptr=str; *ptr;)
            if ((*str=*ptr++)!=c)
                str++;

    *str='\0';

    return ptr-str;
}

void main(void)
{


    char str_arr[]="abcdefghigklmgggnop";
    char ch = 'g';
    int i;
    i = what(str_arr, ch);

    printf("%d\n",i);
}
```

<u>פתרון:</u>

הפונקציה מחזירה את מספר ההכלות של התו c **במחרוזת** str**. בדוגמה שלנו תחזיר הפונקציה** 5**.**

תרגיל 12

מה יהיה הפלט של התכנית הזאת?

```
/*
  output
*/

#include <stdio.h>

void main(void)
{

    int a;

    for(a=3; a>-5; a--) {
            printf(" %d,",a&(a&&a)&a);
            printf(" %d\n",a|(a||a)|a);
    }

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

```
1, 3
0, 3
1, 1
0, 0
1, -1
0, -1
1, -3
0, -3
```

<u>תרגיל 13</u>

מה יהיה הפלט של התכנית הזאת?

```
/*
  output
*/

#include <stdio.h>
#define ADD(A,B)(A+B)

void main(void)
{

    int x=5, y=2, z;

    z = ADD(x, ++y);
    printf("%d %d %d",x,y,z);

}
```

<u>פתרון</u>

הפלט יהיה (הסבירו מדוע!):

5 3 8

תרגיל 14

מה יהיה הפלט של התכנית הזאת?

```
/*
  output
*/

#include <stdio.h>

void main(void)
{

    static int f[2]={1}, i;

    while ((f[i=!i]=f[0]+f[1])<50)
            printf("%d\n",f[i]);

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

```
1
2
3
5
8
13
21
34
```

<u>תרגיל 15</u>

מה יהיה הפלט של התכנית הזאת?

(ניתן להיעזר בסעיף B1.6 בעמוד 248 שבספר הקורס.)

```
/*
  output (see B1.6 on page 248 )
*/

#include <stdio.h>
#define SEEK_SET 0
#define SEEK_CUR 1
#define SEEK_END 2


int main(int argc, char **argv)
{

    FILE *fp = fopen(argv[1], "r");
    fseek(fp, -1L, SEEK_END);
    while (ftell(fp)) {
            putchar(fgetc(fp));
            fseek(fp, -2L, SEEK_CUR);
    }
    putchar(fgetc(fp));

}
```

<u>פתרון</u>

הפלט יהיה:

אם נריץ את התכנית ללא שם קובץ, התכנית "תעוף" (כיצד נתקן זאת?).

אם נריץ את התכנית עם קובץ המכיל:

```
1, 3
0, 3
1, 1
0, 0
1, -1
0, -1
1, -3
0, -3
```

נקבל את הפלט:

```
3- ,0
3- ,1
1- ,0
1- ,1
0 ,0
1 ,1
3 ,0
3 ,1
```

אם נריץ את התכנית עם קובץ המכיל:

```
    1      1   16384    4000
    4      4    4096    1000
   16     10    1024     400
   64     40     256     100
  256    100      64      40
 1024    400      16      10
 4096   1000       4       4
16384   4000       1       1
65536  10000       0       0
```

נקבל את הפלט:

```
0      0      00001   63556
1      1      0004    48361
4      4      0001    6904
01     61     004     4201
04     46     001     652
001    652    04      46
004    4201   01      61
0001   6904   4       4
0004   48361  1       1
```

כלומר, התכנית קוראת קובץ, והפלט הוא הדפסתו מן הסוף להתחלה.

תרגיל 16

מה יהיה הפלט של התכנית הזאת?

```
/*
  find output
*/

#include <stdio.h>

void main(void)
{

    unsigned char i,m=0xFF,n=0x1;

    for(i=0; i<=8; i++,n+=n,m/=2)
          printf("%5x%5x%5x%5x%5x%5x\n",n,m,n&m,n|m,n^m,~n);


}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

```
  1   ff   1   ff   fefffffffe
  2   7f   2   7f   7dfffffffd
  4   3f   4   3f   3bfffffffb
  8   1f   8   1f   17fffffff7
 10    f   0   1f   1ffffffef
 20    7   0   27   27ffffffdf
 40    3   0   43   43ffffffbf
 80    1   0   81   81ffffff7f
  0    0   0    0   0ffffffff
```

תרגיל 17

מה יהיה הפלט של התכנית הזאת?

```
/*
  find output
*/

#include <stdio.h>

void main(void)
{

    union {
    int index;
    struct {
            unsigned int x:2;
            unsigned int y:1;
            unsigned int z:1;
    } bits;
    } number;

    for(number.index=3; number.index<8; number.index++)
            printf("%3d\t%3d%3d%3d\n",number.index, number.bits.x,
                    number.bits.y, number.bits.z);

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

```
 3   3  0  0
 4   0  1  0
 5   1  1  0
 6   2  1  0
 7   3  1  0
```

תרגיל 18

מה יהיה הפלט של התכנית הזאת?

```
/*
  find output
*/

#include <stdio.h>

void main(void)
{

    int small=1, big=0x4000, index;

    for(index=0; index<9; index++,small<<=2,big>>=2)
          printf("%8d%8x%8d%8x\n",small, small, big, big);

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

```
       1       1   16384    4000
       4       4    4096    1000
      16      10    1024     400
      64      40     256     100
     256     100      64      40
    1024     400      16      10
    4096    1000       4       4
   16384    4000       1       1
   65536   10000       0       0
```

<u>תרגיל 19</u>

מה יהיה ערכו של המשתנה j לאחר ביצוע השורות האלה?

```
/*
  find j
*/

#include <stdio.h>
#define SQUARE(X) ((X)*(X))

void main(void)
{
     int j, i=3;
     j = SQUARE(++i);
}
```

<u>פתרון</u>

הערך יהיה 25 (הסבירו מדוע!).

תרגיל 20

כמה פעמים יודפס המשפט "Hello World" לאחר הרצת התכנית הבאה:

א. אם התכנית תורץ עם הפרמטרים 4 9 85 ?

ב. אם התכנית תורץ עם הפרמטרים 4 9 ?

ג. אם התכנית תורץ עם הפרמטר 4?

```
/*
  find "Hello World"
*/

#include <stdio.h>

void main(int argc, char *argv[])
{

    int i;
    if (argc > 2)
        for(i=0; i<atoi(argv[argc-1]); i++)
            main(argc-1, argv);
    else printf("Hello World\n");

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

א. אם התכנית תורץ עם הפרמטרים 4 9 85 המשפט יודפס 765 פעמים.

ב. אם התכנית תורץ עם הפרמטרים 4 9 המשפט יודפס 9 פעמים.

ג. אם התכנית תורץ עם הפרמטר 4 המשפט יודפס פעם אחת.

<u>תרגיל 21</u>

מה יהיה הפלט של התכנית הזאת?

```
/*
  find output
*/

#include <stdio.h>
#include <string.h>
void main(void)
{

    char array[6], table[2][5]={'a','b','c','d','\0','f','g','h','i','\0'};
    strcpy(array, &table[1][0]);
    printf("%s\n%s\n", table, array);

}
```

<u>פתרון</u>

הפלט יהיה (הסבירו מדוע!):

```
abcd
fghi
```

תרגיל 22

מדוע ההצבה *p=8 אינה חוקית בתחילת התכנית הבאה (ההצבה מופיעה כהערה):

```
/*
  find out why can't we use *p=8
*/

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

void main(void)
{

    int memallocated;
    int *p, *q;

    //*p = 8;

    p = &memallocated;
    *p = 8;
    printf("%d %d\n",*p, memallocated);

    q = (int *)malloc(sizeof(int));
    *q = 8;
    printf("%d \n",*q);

}
```

פתרון

ניתן להציב למצביע רק לאחר שקיבל כתובת חוקית, שהוקצתה לתכנית (כפי שבוצע בשני המקרים שבהמשך).

<u>תרגיל 23</u>

מה שגוי בתכנית הזאת?

```
/*
  use string and pointer
*/

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

void main(void)
{

    char *s;

    s = (char *)malloc(15);
    s = "Hello World";
    free(s);

}
```

<u>פתרון</u>

התכנית "תעוף" בשל השורה "free(s)" . **(הסבירו מדוע!).**

תרגיל 24

מה יהיה הפלט של התכנית הזאת?

```
/*
  array of pointers
*/

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

void main(void)
{

    char *name[] = {"judy","mike","david","cochavit"};

    printf("%c\n", **name);
    printf("%s\n", *name);
    printf("%s\n", *(name+1));
    printf("%s\n", name[2]);
    printf("%c\n", *(*(name+1)+2));
    printf("%c\n", **(name+1));
    printf("%c\n", **(name+1)+7);

}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

```
j
judy
mike
david
k
m
t
```

תרגיל 25

מה יהיה הפלט של התכנית הבאה. יש להראות דוגמה עם מספר ארגומנטים לתכנית:

```
/*
  arguments
*/

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

void main(int argc, char *argv[])
{

    if (argc == 1)
            printf("no arguments only program name");
    else
            while (--argc)
                printf("%s \n", *++argv);

}
```

פתרון

אם נריץ עם הארגומנטים: aaa bbb ccc ddd **הפלט יהיה (הסבירו מדוע!):**

```
aaa
bbb
ccc
ddd
```

תרגיל 26

מה עושה התכנית הבאה, ומה יהיה הפלט שלה?

```
/*
  find output
*/

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

void main(void)
{

    char s[80];
    char *comma=",";
    char *cp;
    gets(s);
    printf("%s\n",s);
    cp=strtok(s,comma);
    printf("%s\n",cp);
    while(cp=strtok(NULL,comma))
            printf("%s\n",cp);

}
```

פתרון

התכנית מדגימה שימוש ב-strtok **:**

התכנית קוראת מחרוזת מ-stdin **על ידי** gets() **על ידי שימוש ב-**strtok **מודפסת המחרוזת, בשורות נפרדות, כאשר הסימן ",," מהווה המפריד בין המילים להדפסה בשורות הנפרדות.**

לדוגמה, אם נכתוב: aaa,bbb,ccc,ddd **יודפס:**

aaa
bbb
ccc
ddd

<u>תרגיל נוסף:</u> יש לקרוא ב-man, מדוע השימוש ב-gets() **אינו מומלץ, ולהראות שימוש בפונקציית ספרייה חלופית.**

תרגיל 27

מה עושה התכנית הבאה, ומה יהיה הפלט שלה?

```
/*
Bubble sort – pointers to functions
*/

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

void bubble_sort(int array[], int number_of_elements, int(*cmp)(int,int));
int ascending(int i,int j);
int descending(int i,int j);

#define LENGTH 20


void main(void)
{
   int i,
   numbers[LENGTH]={60,6,4,5,18,180,7,9,66,87,120,18,18,67,17,18,19,0,200,0};
   printf("The unordered version of numbers is:\n");
   for(i=0;i<LENGTH;i++)
        printf("%d ",numbers[i]);
   putchar('\n');

   bubble_sort(numbers, LENGTH, ascending);

   printf("The ordered ascending version of numbers is:\n");
   for(i=0;i<LENGTH;i++)
        printf("%d ",numbers[i]);
   putchar('\n');

   bubble_sort(numbers, LENGTH, descending);
```

```
    printf("The ordered descending version of numbers is:\n");
    for(i=0;i<LENGTH;i++)
        printf("%d ",numbers[i]);
    putchar('\n');

}

int ascending(int i,int j)
{
    return i-j;
}
int descending(int i,int j)
{
    return j-i;
}


/* Bubble sort */
void bubble_sort(int array[], int number_of_elements, int(*cmp)(int,int))
{
    int n,i,temp;
    for (;number_of_elements;number_of_elements--)
        for(i=0;i<number_of_elements-1;i++)
        if ((*cmp)(array[i],array[i+1])>0){
            temp = array[i];
            array[i] = array[i+1];
            array[i+1] = temp;
        }


}
```

<u>פתרון</u>

התכנית מדגימה מיון בועות כאשר הפונקציה bubble_sort **מקבלת מערך למיון, גודל מערך, ומצביע לפונקציה, אשר מקבלת שני מספרים שלמים ומחזירה מספר שלם.**

כך הפונקציות ascending **ו-**descending **יועברו לפונקציה** bubble_sort **בהתאם לסדר המיון הרצוי (עולה או יורד).**

התכנית מכילה מערך קבוע. פלט התכנית יהיה:

```
The unordered version of numbers is:
60 6 4 5 18 180 7 9 66 87 120 18 18 67 17 18 19 0 200 0
The ordered ascending version of numbers is:
0 0 4 5 6 7 9 17 18 18 18 18 19 60 66 67 87 120 180 200
The ordered descending version of numbers is:
200 180 120 87 67 66 60 19 18 18 18 18 17 9 7 6 5 4 0 0
```

תרגיל 28

מה יהיה הפלט של התכנית הזאת? (יש לשים לב לסדר הקדימויות.)

```
/*
find output
*/

#include <stdio.h>

#define LENGTH 10

struct con{
    int num;
    char *word;
};

void printarray(struct con fuse[]);

void main(void)
{
    struct con fuse[]={
    {10,"ten"},
    {20,"twenty"},
    {30,"thirty"},
    {40,"forty"},
    {50,"fifty"},
    {60,"sixty"},
    {70,"seventy"},
    {80,"eighty"},
    {90,"ninety"},
    {100,"one hundred"}
    };

    struct con *p;
```

```
    p = fuse;

    ++p->num; /* equal to ++(p->num) increments num */
    printarray(fuse);

    printf("%d\n",(++p)->num); /* increments p before accessing num */
    printf("%d\n",p->num);
    printf("%d\n",(p++)->num); /* increments p after accessing num. () 
unnecessary */
    printf("%d\n",p->num);
    printarray(fuse);

    printf("%d\n",p->num++); /* increments num after accessing it */
    printf("%d\n",p->num);
    printarray(fuse);

    printf("%s\n",p->word);
    printf("%c\n",*p->word); /* what word points to */
    printf("%c\n",*p->word++); /* word increments after accessing what it points 
to */
    printarray(fuse);

    // (*p->word)++; undefined, tries to access string contant. increments what 
word points to
    printarray(fuse);

    printf("%c\n",*p++->word); /* increments p after accessing what word points 
to */
    printf("%c\n",*p->word);
    printf("%c\n",*++p->word); /* increments word then accesses what it points to 
*/
    printf("%c\n",*p->word);
    // printf("%c\n",++(*p->word));  undefined, increments what word points to
    printf("%c\n",(*p->word)+1); /* what word points to + 1 */
```

```
    printarray(fuse);

}

void printarray(struct con fuse[])
{
    int i;
    for(i=0;i<LENGTH;i++)
        printf("%d\t%s\n",fuse[i].num,fuse[i].word);

    putchar('\n');
}
```

<u>פתרון</u>

הפלט יהיה (הסבירו מדוע!):

```
11 ten
20 twenty
30 thirty
40 forty
50 fifty
60 sixty
70 seventy
80 eighty
90 ninety
100         one hundred

20
20
20
30
11 ten
20 twenty
30 thirty
```

40 forty
50 fifty
60 sixty
70 seventy
80 eighty
90 ninety
100 one hundred

30
31
11 ten
20 twenty
31 thirty
40 forty
50 fifty
60 sixty
70 seventy
80 eighty
90 ninety
100 one hundred

thirty
t
t
11 ten
20 twenty
31 hirty
40 forty
50 fifty
60 sixty
70 seventy
80 eighty
90 ninety
100 one hundred

11 ten
20 twenty
31 hirty
40 forty
50 fifty
60 sixty
70 seventy
80 eighty
90 ninety
100 one hundred

h
f
o
o
p
11 ten
20 twenty
31 hirty
40 orty
50 fifty
60 sixty
70 seventy
80 eighty
90 ninety
100 one hundred

תרגיל 29

נתונה תכנית הכוללת פונקציה, בעלת מספר משתנה של ארגומנטים. מה יהיה פלט התכנית?

```
/*
  arguments
*/

#include <stdarg.h>
#include <stdio.h>

int what(int firstint,...)
{
    int val = -999; /* the end of the list of int is marked by -999*/
    int x = 0;
    va_list argp;
    va_start(argp, firstint);
    x = firstint;
    while (x != -999) {
          if (val < x) val = x;
          x = va_arg(argp, int);
    }
    va_end(argp);
    return(val);

}

void main(void)
{
    int i;

    i = what(-1,20,30,-40,-999);
    printf("%d\n",i);
```

```
    i = what(10,9,8,7,6,5,4,3,2,1,-999);
    printf("%d\n",i);

}
```

<u>פתרון</u>

הפלט יהיה (הסבירו מדוע!):

30
10

תרגיל 30

התכנית הבאה מדגימה עבודה עם קבצים. מה יהיה פלט התכנית?

```
/*
  file positions
*/

#include <stdio.h>

#define LENGTH 40

void main(void)
{
    FILE *iofp;
    long pos;
    int c;
    char s[LENGTH];

    /* create binary file for update discard previous contents if any */
    if ((iofp=fopen("ioexample", "w+b ")) == NULL)
        printf("Can't open file");
    else {
            fprintf(iofp, "this is being printed to the file");
            pos = ftell(iofp);
            printf("current position %d\n", pos);

            rewind(iofp);

            pos = ftell(iofp);
            printf("current position %d\n", pos);

            c = fgetc(iofp);
            printf("char read is %c\n", c);
            pos = ftell(iofp);
            printf("current position %d\n", pos);

            /* in between reading and writing must use a file positioning function: */
```

```
        fseek(iofp, 0, SEEK_CUR);

        c = fputc('z',iofp);
        printf("%c\n", c);
        rewind(iofp);

        pos = ftell(iofp);
        printf("current position %d\n", pos);

        c = fgetc(iofp);
        printf("char read is %c\n", c);
        pos = ftell(iofp);
        printf("current position %d\n", pos);
        c = fgetc(iofp);
        printf("char read is %c\n", c);
        pos = ftell(iofp);
        printf("current position %d\n", pos);

        rewind(iofp);
        fseek(iofp, 2, SEEK_SET);
        pos = ftell(iofp);
        printf("current position %d\n", pos);
        c = fgetc(iofp);
        printf("char read is %c\n", c);
        rewind(iofp);
        fgets(s,LENGTH,iofp);
        printf("%s\n", s);

        printf("that is all");
        fclose(iofp);

    }
}
```

פתרון

בקובץ ioexample **ייכתב:** tzis is being printed to the file

פלט התכנית יהיה (הסבירו מדוע!):

```
current position 33
current position 0
char read is t
current position 1
z
current position 0
char read is t
current position 1
char read is z
current position 2
current position 2
char read is i
tzis is being printed to the file
that is all
```

תרגיל 31

התכנית הבאה מדגימה קריאה וכתיבה של מבנים לקבצים. מה יהיה פלט התכנית?

```
/*
  file:  read/write struct
*/

#include <stdio.h>

#define LENGTH 4

void main(void)
{
    FILE *iofp;
    int i;

    struct r {
    int x;
    char c;
    } st[LENGTH] = {0,'0',1,'1',2,'2',3,'3'};

    struct r temp;


    /* open binary file for update discard previous contents if any */
    if ((iofp=fopen("ioexample", "w+b ")) == NULL)
          printf("Can't open file");
    else {
          fwrite(&st, sizeof(struct r),3,iofp);
          fseek(iofp, sizeof(struct r)*2, SEEK_SET);
          fread(&temp, sizeof(struct r),1,iofp);

          printf("%d,%c\n", temp.x,temp.c);
```

```
        temp.x = 99;
        temp.c = 'z';

        fseek(iofp, sizeof(struct r)*2, SEEK_SET);
        fwrite(&temp, sizeof(struct r),1,iofp);

        rewind(iofp);

        fread(&st, sizeof(struct r),3,iofp);
        for(i=0;i<LENGTH;i++)
            printf("%d,%c\n", st[i].x,st[i].c);

        fclose(iofp);
    }
}
```

פתרון

הפלט יהיה (הסבירו מדוע!):

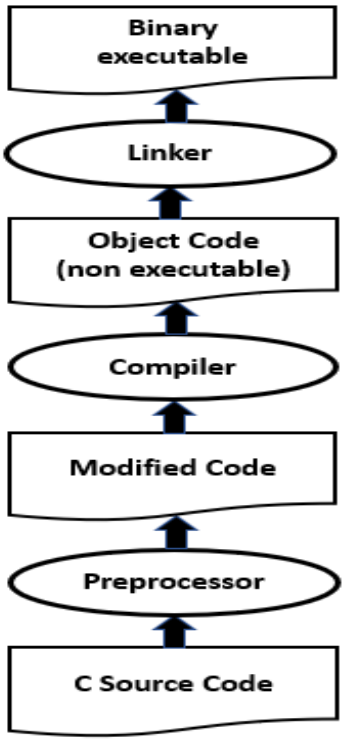
```
2,2
0,0
1,1
99,z
3,3
```

נספח ג

תכניות מאקרו

שלב קדם המעבד – C Preprocessor

שלב קדם המעבד (C Preprocessor) הוא שלב עצמאי בתהליך ההידור. שלב זה מהווה בעיקר כלי להחלפת מלל (טקסט), המכין את נתוני השפה לקראת ההידור. פקודות קדם המעבד יכולות להופיע בכל מקום בקוד, והן מוגדרות באמצעות התו "#".



תפקידיו העיקריים של קדם המעבד:

1. הגדרות קבועים
2. הכללת קבצים:

   ההוראה "#include" מאפשרת לכלול קובצי כותרת בתכנית.

   שמות קבצים אלה מסתיימים ב-"h." וכתובים בפורמט "xxx.h".

   ההוראה #include "abc.h" כוללת קובצי כותרת של הגדרות המשתמש, וההוראה #include <abc.h> כוללת קובצי כותרת של המערכת.
3. אוסף תנאים: מאפשר הפעלת קוד במצבים או בפלטפורמות ספציפיות בלבד.
4. הרחבת מאקרו: החלפת פרמטר בטקסט ללא בדיקת סוג (type checking). הקוד עשוי לפעול מהר יותר.

הוראות נפוצות

תחביר	הוראה	תיאור
#define <id> <text>	#define XXX	הגדרת קבועים
#define <id> (<pram list>)	#define XXX (P1,P2,P3)	הגדרת מקרואים
#undef <id>	#undef XXX	ביטול הגדרת קבועים / מקרואים
#include <<file>>	#include <XXX.h>	הכללת קובצי כותרת בתכנית
#include "<file>"	#include "XXX.h"	הכללת קובץ שנמצא במדריך הנוכחי
#include_next "<file>"	#include <XXX.h>	להכללת הקובץ הבא באותו שם GNU
#import "<file>"	#import "XXX.h"	
#ifdef <id>	#ifdef XXX	מחזיר ערך חיובי אם המאקרו מוגדר
#ifndef <id>	#ifndef XXX	מחזיר ערך חיובי אם המאקרו אינו מוגדר
#if <expression> <text> #endif	#if #if (XXX == YYY) : #elif (!XXX && YYY) : #elif (XXX || YYY) : #else : #endif	בדיקה אם התנאי מתקיים (בזמן ריצה) Check if condition is true (compile time) operators: == (equality) != (inequality) && (and) || (or) !
	#else	מבוצע רק אם לא חוזר ערך חיובי מהתנאי (ההוראה המותנית)
	#elif	elseIf
	#endif	מציין את סוף ההוראה המותנית
	#line	מאפשר לשנות את מספר השורה של המהדר ואת שם הקובץ של השגיאות והאזהרות
	#error	מאפשר ליצור שגיאה ממקום מסוים בקוד
	#warning	מאפשר ליצור אזהרה ממקום מסוים בקוד

דוגמאות

1) include

```
#include <stdio.h>
#include "myFile.h"
    /* gets stdio.h from System Libraries and add the text to the current source. */
    /* gets myheader.h from the local directory and add the content to the current source */
```

2) define

```
#define ARRAY_SIZE 100
    /* replace instances of ARRAY_SIZE in the program with 100. */
    /* commonly used for declaring constants in c to increase readability. */
```

3) undef

```
#undef ARRAY_SIZE
#define ARRAY_SIZE 1000
    /* un-define the existing ARRAY_SIZE and re-define it with size = 1000*/
```

4) ifndef

```
#ifndef PI
#define PI 3.141593
#endif
    /* define PI only if PI is not defined */
#ifndef NULL
#define NULL (void *)0
#endif
```

5) ifdef

```
#ifdef PI
#undef PI
#endif
    /* Un-define PI only if PI exists */
```

האופרטור Defined()

באופרטור זה משתמשים כדי לקבוע אם כבר הוגדר הסמל לקבוע.

```
#include <stdio.h>

#if !defined (MESSAGE)
   #define MESSAGE "This is a C program"
   #endif

int main(void) { printf("%s\n", MESSAGE);
   return 0;
}
```

Output:

This is a C program

כיצד להימנע מהכללת קבצים מרובים

*f1.h

```
int sum(int a, int b)
{
   return a+b;
}
```

*f2.h

```
#include "f1.h"
int mult(int a, int b)
{
   return a*b;
}
```

main.c

```
#include "f2.h"
#include "f1.h"
int main()
{
   return 0;
}
```

```
# 1 "main.c"
# 1 "<built-in>"
# 1 "<command- line>"
# 1 "main.c"
# 1 "f2.h" 1
# 1 "f1.h" 1
int sum(int a, int b) { return
   a+b;
}
# 2 "f2.h" 2
int mult(int a, intb) {
   return a*b;
}
# 2 "main.c" 2
# 1 "f1.h" 1
int sum(int a, int b) { return
   a+b;
}
# 3 "main.c" 2 int
main() { return 0;
}
```

f1.h

```
#if !defined(__F1__HEADER__)
#define__F1__HEADER__
int sum(int a, int b) {
return a+b;
}
#endif
```

f2.h

```
#include "f1.h"
int mult(int a, int b)
{
   return a*b;
}
```

main.c

```
#include "f2.h"
#include "f1.h"
int main()
{
   return 0;
}
```

```
# 1 "main.c"
# 1 "<built-in>"
# 1 "<command- line>"
# 1 "main.c"
# 1 "f2.h" 1
# 1 "f1.h" 1
int sum(int a, int b) { return
   a+b;
}
# 2 "f2.h" 2
int mult(int a, intb) {
   return a*b;
}
# 2 "main.c" 2

int main() {
   return 0;
}
```

*הערה: לא נהוג לממש פונקציות בקובצי כותרת

כיצד לדלג על קוד בזמן קומפילציה

```
#include <stdio.h> void
printArr(int*, int); int main( void )
{
  int i;
  int arr[] = {1 , 2 , 3 , 4 , 5};
  int temp = arr[4];

  #if DEBUG
   printArr(arr, 5); #endif

  for(i=4 ; i>0 ; i--) {
    arr[i] = arr[i-1];
  }
  arr[0] = temp;

  #if DEBUG
   printArr(arr, 5);
  #endif

  return 0;
}

void printArr(int * pArr, int size) {
  int i;
  for(i=0; i<size ; i++)
     printf("%d " ,
     pArr[i]);
  putchar('\n');
}
```

```
student@ubuntu:~/Desktop/Y2019March/mmn$ gcc -ansi -Wall -pedantic m.c
student@ubuntu:~/Desktop/Y2019March/mmn$ ./a.out
student@ubuntu:~/Desktop/Y2019March/mmn$ gcc -ansi -Wall -pedantic m.c -DDEBUG
student@ubuntu:~/Desktop/Y2019March/mmn$ ./a.out
1 2 3 4 5
5 1 2 3 4
student@ubuntu:~/Desktop/Y2019March/mmn$
```

```
student@ubuntu:~/Desktop/Y2019March/mmn$ gcc -E m.c -DDEBU
# 1 "m.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "m.c"
void printArr(int*, int);
int main( void ) {
   int i;
   int arr[] = {1 , 2 , 3 , 4 , 5};
   int temp = arr[4];

   printArr(arr, 5);

   for(i=4 ; i>0 ; i--) {
     arr[i] = arr[i-1];
   }
   arr[0] = temp;

   printArr(arr, 5);

   return 0;
}
```

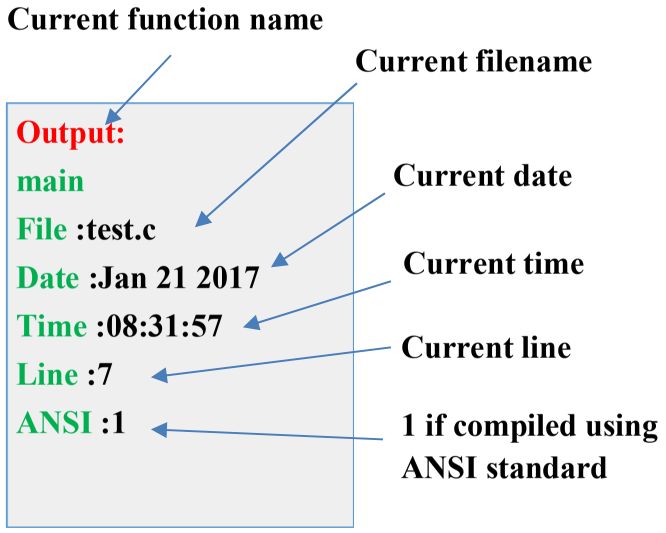
מאקרו מוגדרים מראש – Predefined Macros

- Standard predefined macros e.g.___FILE___
- Common Predefined Macros e.g. ___COUNTER___
- System-specific Predefined Macros

test.c

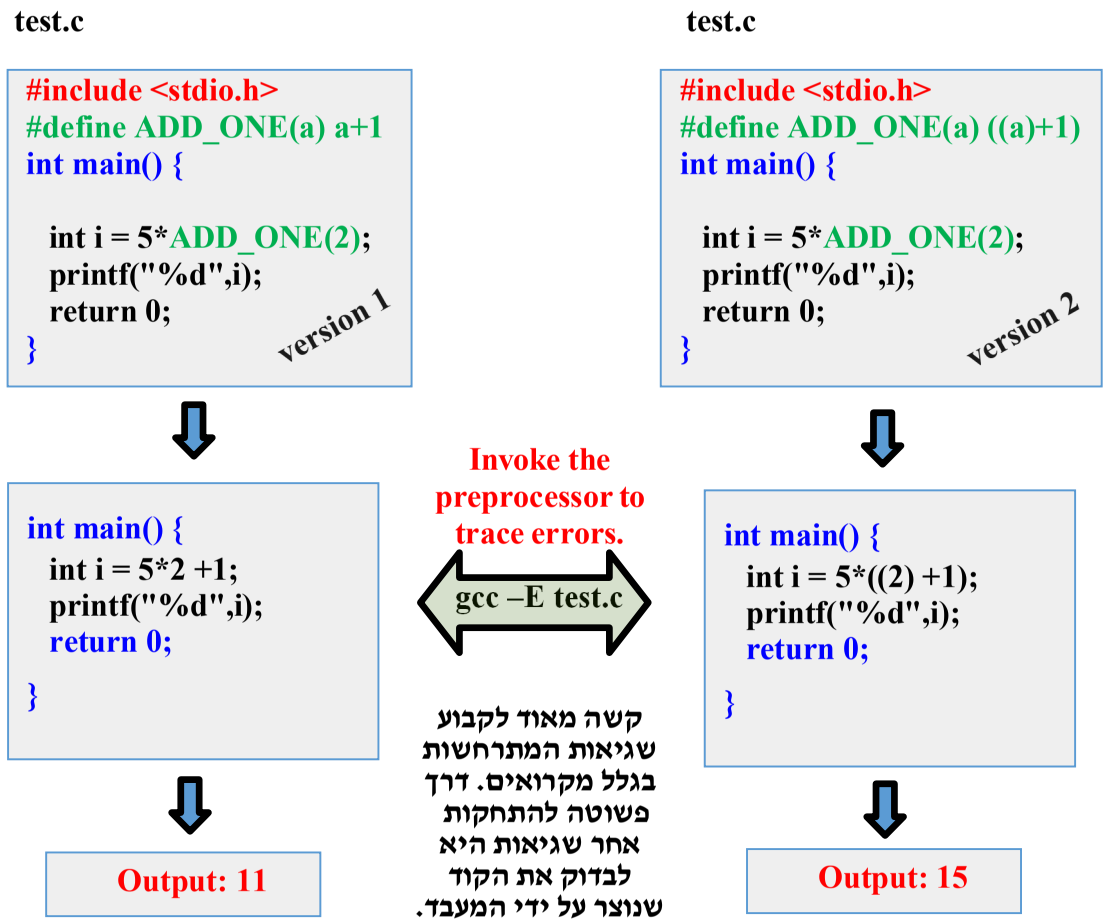
```
#include <stdio.h>
int main() {

  printf("File :%s\n",___func___);
  printf("File :%s\n",___FILE___);
  printf("Date :%s\n",___DATE___);
  printf("Time :%s\n",___TIME___);
  printf("Line :%d\n",___LINE___);
  printf("ANSI :%d\n",___STDC___);


return 0;
}
```

standard predefined macros



דוגמאות מאקרו נוספות

דוגמה 1



דוגמה 2

```
#include <stdio.h>
#define MAX(a,b) a>b? a:b
int main() {
  int a=1,b=2;

int x=MAX(a, b);
printf("X:%d\n", x);

int y=MAX(a+1,b+1);
printf("Y:%d\n", y);

  int z=MAX(a+5,b+1);
  printf("Z:%d\n", z); return 0;
}
```

```
int main() {
  int a=1,b=2;

  int x=a>b? a:b;
  printf("X:%d\n", x);

  int y=a+1>b+1? a+1:b+1;
  printf("Y:%d\n", y);

  int z=a+5>b+1? a+5:b+1;
  printf("Z:%d\n", z);
  return 0;
}
```

Output:

X:2→ 1>2?1:2 → 2
Y:3→ 1+1>2+1?1+1:2+1 → 2+1 → 3
Z:6→ 1+5>2+1?1+5:2+1 → 5+1 → 6

המשך דוגמה 2

```
#include <stdio.h>

#define MAX(a,b) (a)^(((a)^(b))&-((a)<(b)))

#define MAX1(a,b) a>b?a:b

#define MAX2(a,b) (((a)>(b))?(a):(b))

#define MAX3(a,b) \
  ({ typeof (a) _a = (a); \
     typeof (b) _b = (b); \
     _a>_b?_a:_b; })

#define MAX4(a,b) ( \
   { __auto_type __a = (a);
     __auto_type __b = (b); \
     __a>__b?__a:__b; })
```

דוגמה 3

```
#include <stdio.h>
#define MULT(a, b) a*b
int main(void) {
    int x = MULT(3,4);
    printf("X:%d\n", x);
    int y = MULT(3+2,4+2);
    printf("Y:%d\n", y);
    return 0;
}
```

version 1

Output:
X:12
Y:13 → 3+2*4+2 →13;

```
#include <stdio.h>
#define MULT(a, b) (a)*(b)
int main(void) {
    int x = MULT(3,4);
    printf("X:%d\n", x);
    int y = MULT(3+2,4+2);
    printf("Y:%d\n", y);
    return 0;
}
```

version 2

Output:
X:12
Y:30 → (3+2)*(4+2) → 30

המרת כל סוג למחרוזת תווים (String)

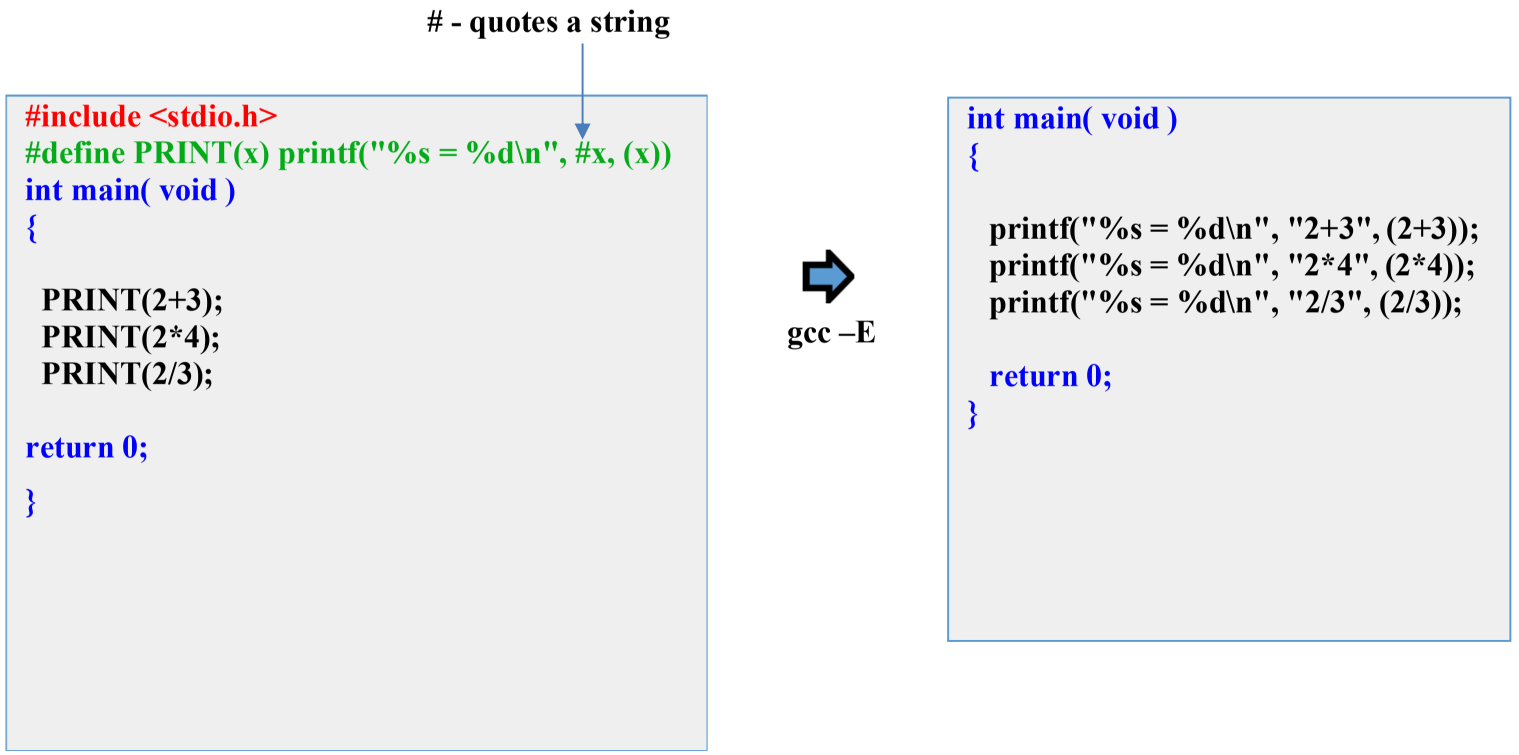
- quotes a string

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

#define STRING(x) #x
int main()
{
char* s1 = STRING(abc ef);
char* s2 = STRING(123);
char* s3 = STRING(1.5);

  printf("%s\n", s1);
  printf("%s\n", s2);
  printf("%s\n", s3);

  return 1;
}
```

gcc –E

```
int main()
{
  char* s1 = "abc ef"; char*
  s2 = "123"; char* s3 =
  "1.5";

  printf("%s\n", s1);
  printf("%s\n", s2);
  printf("%s\n", s3);

  return 1;
}
```

הדפסת ביטוי



שרשור

is used for concatenation

```
#include <stdio.h>
#include <string.h>
#define CONCAT(x,y)\
            STRING(x ## y)
#define STRING(s) #s

int main()

{

 char* s1 =CONCAT(abc,edf);
 char* s2 = CONCAT(abc,123);
 char* s3 = CONCAT(123,4.5);

 printf("%s\n", s1);
 printf("%s\n", s2);
 printf("%s\n", s3);


 return 1;
}
```

gcc –E

```
int main()
{
  char* s1 = "abcedf";
  char* s2 = "abc123";
  char* s3 = "1234.5";

  printf("%s\n", s1);
  printf("%s\n", s2);
  printf("%s\n", s3);

  return 1;
}
```

הקצאה כללית של מערך (Block)

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

#define BLOCK_ALLOCATION(pointer , type, size) \
pointer=(type*) malloc (sizeof(type) * size)

int main( void )
{
  char* pc = NULL;
  int *px = NULL;
  float *py = NULL;

  BLOCK_ALLOCATION(pc, char, 3);
  BLOCK_ALLOCATION(px, int, 3);
  BLOCK_ALLOCATION(py, float, 3);

  return 0;
}
```

gcc –E

```
int main( void )

{
char* pc = ((void *)0);
int *px = ((void *)0);
float *py = ((void *)0);

pc=(char *) malloc (sizeof( char) * 3);
px=(int *) malloc (sizeof( int) * 3);
py=(float *) malloc (sizeof( float) * 3);


  return 0;

}
```

Swap כללי

```
#include <stdio.h>

#define SWAP(a, b, type) \
    { \
    type tmp = a;\
    a=b;\ b=tmp;\
    }

int main(void ) {
  char c1='a' , c2='b';
  int x1=2 , x2=3;
  double y1=2.5 , y2 =3.5;
  int* px1 = &x1, px2 = &x2;

  SWAP(c1,c2,char);
  SWAP(x1,x2,int);
  SWAP(y1,y2,double);
   SWAP(px1,px2,int*);

  return 0;
}
```

gcc –E

```
int main( void ) {
  char c1='a' , c2='b';
  int x1=2 , x2=3;
  double y1=2.5 , y2 =3.5;
  int* px1 = &x1, px2 = &x2;

  { char tmp = c1; c1=c2; c2=tmp; };
  { int tmp = x1; x1=x2; x2=tmp; };
  { double tmp = y1; y1=y2; y2=tmp; };
  { int* tmp = px1; px1=px2; px2=tmp; };

  return 0;
}
```

המשך

```
#include <stdio.h>

#define SWAP1(a, b, type)\
      { \
          type tmp = a;\
          a=b;\ b=tmp;\
      }

 #define SWAP2(a, b, type)\
   do { type tmp = a;\
        a = b;\
        b = tmp;\
   } while (0)

#define SWAP3(a, b, type)\
  do { typeof(a) tmp = a;\
       a = b;\
       b = tmp;\
  } while (0)
```

```
#include <stdio.h>

#define SWAP4(a,b) do \
  { unsigned char\
    tmp[sizeof(a) == sizeof(b)?\
    (signed)sizeof(a) : -1];\
    memcpy(tmp,&b,sizeof(a));\
    memcpy(&b,&a,sizeof(a));\
    memcpy(&a,tmp,sizeof(a));\
   } while(0)
```

נספח ד

מדריך לסביבת UNIX

תוכן העניינים

1. הקדמה

UNIX היא מערכת הפעלה אשר פותחה במקור על ידי קן תומפסון ודניס ריצ'י, בשנות הששים והשבעים של המאה ה-20. הפיתוח החל במעבדות הטלפוניה בל, בארצות הברית (AT&T Bell Labs). כיום מערכות UNIX (או מערכות "דמויות UNIX", כלומר כאלה המכילות את רוב המרכיבים של המערכת) נתמכות על ידי גופים רבים, מתוכם גם גופים שלא למטרות רווח. במקרים אלו תהיה זו בדרך כלל "מערכת פתוחה", כלומר ניתן לקבל את המערכת ואת הקוד שבו נכתבה ללא תשלום; כך שהמערכת זמינה לשימוש, לשינוי ולהפצה לכל דורש.

המערכת התואמת לחלוטין להגדרות המקוריות של UNIX נתמכת כיום על ידי קבוצת SCO ונובל (Novell) בארצות הברית. אחד הכלים החשובים של מערכת UNIX הוא שפת C, אשר שימשה לכתיבת מערכת ההפעלה כבר בזמן הפיתוח הראשוני. UNIX תוכננה להיות ניידת, כלומר בלתי תלויה בפלטפורמה (portablility), ולאפשר ריבוי תהליכים וריבוי משתמשים בו זמנית.

בקורס "מעבדה בתכנות מערכות" נשתמש במערכת הפעלה פתוחה דמוית UNIX.

הקורס כולל תרגול רב כדי לסייע לכם בלימוד סביבת העבודה.

נפרסם בכל סמסטר הנחיות אשר מטרתן להסביר את הנקודות החשובות לגבי סביבת העבודה, לגבי ההתקנה ולגבי סביבת התכנות (המעבד, המהדר וכו').

2. כיצד מתחילים

בכל סמסטר נפרסם כיצד להתקין את מערכת ההפעלה העדכנית.

2.1 התקנה

- יש לוודא שהמחשב שבו מתקינים את מערכת ההפעלה אכן מקיים את דרישות החומרה המופיעות בהוראות ההתקנה.
- במידת הצורך עליכם לשנות את הרשאת הקבצים מ-"read only" ל-"read/write".

2.2 הפעלה

ההפעלה תתאפשר לאחר ששלב ההתקנה הסתיים ללא תקלות.

ישנה חשיבות מרובה ליציאה מסודרת מהמערכת. **אין לסגור** את המערכת על ידי לחיצת עכבר בפינת החלון, כפי שסוגרים חלון רגיל במערכת הפעלה Windows.

3. סביבת תכנות

3.1 בחירת סביבת עבודה

בכל סמסטר נפרסם תיאור מפורט של המעבדים האפשריים, בחירת סביבה (shell), צורת הקומפילציה ועוד. יש לבחור את המעבד ואת סביבת העבודה על פי ההנחיות המפורסמות בתחילת הסמסטר.

3.2 כיצד להעביר הידור ובניית קובץ ריצה

נשתמש במהדר gcc. הסבר מפורט על השימוש במהדר ניתן בחומר שיפורסם. ניתן לעבוד בחלון shell ולבצע את ההידור, כפי שמופיע בהסבר.
לאחר שעקרונות ההידור ובניית קובץ הריצה הובנו, יש לכתוב קובץ MAKEFILE המכיל את כל הוראות ההידור, הקישור ובניית קובץ הריצה. קובץ ה-MAKEFILE יכול לשמש להידור ולבניית קבצים נוספים בעתיד. יש לקרוא בחומר שיפורסם כדי ללמוד לכתוב קובץ זה.

דוגמה פשוטה לתוכן קובץ MAKEFILE, עבור הדוגמה הראשונה מפרק 9:

```
sample: sample.c
gcc -Wall -g sample.c -o sample

clean:
rm -f sample
```

הסבר לדוגמה: במקרה זה התוכנית כולה מופיעה בקובץ אחד (sample.c), ללא קובצי h. כך תוכן הקובץ הוא פשוט, ומכיל קריאה למהדר gcc, יצירת קובץ ריצה בשם sample, קריאה להצגת אזהרות המהדר על ידי Wall-, וקישור אל הספריות של ה-Debugger על ידי g–.

כדי לקבל הסבר מפורט לכל אלו, וכן פירוט של קבצי MAKEFILE מורכבים יותר, קראו בחומר שנפרסם בתחילת הסמסטר.

תקציר פקודות UNIX

פקודות שליטה בסביבה - Environment Control

פקודה	תיאור
cd d	Change to directory d
mkdir d	Create new directory d
rmdir d	Remove directory d
mv f1 [f2...] d	Move file f to directory d
mv d1 d2	Rename directory d1 as d2
passwd	Change password
alias name1 name2 (csh/tcsh)	Create command alias
alias name1="name2" (ksh/bash)	Create command alias
unalias name1[na2...]	Remove command alias na
exit	End terminal session
setenv name v (csh/tcsh)	Set env var to value v
export name="v" value v (ksh/bash)	set environment variable to

Output, Communication, & Help -

פלט תקשורת ועזרה

פקודה	תיאור
lpr -P printer f *or* lp -d printer f	Output file f to line printer
script [f]	Save terminal session to f
exit	Stop saving terminal session
mailx username	Send mail to user
man name	Unix manual entry for name

פקודות שליטה על תהליכים – Process Control

פקודה	תיאור
CTRL/c *	Interrupt processes
CTRL/s *	Stop screen scrolling
CTRL/q *	Resume screen output
sleep n	Sleep for n seconds
jobs	Print list of jobs
kill %	Kill job n
ps	Print process status stats
kill -9 n	Remove process n
CTRL/z *	Suspend current process
stop %n	Suspend background job n
cmmd&	Run cmmd in background
bg [%n]	Resume background job n
fg [%n]	Resume foreground job n
exit	Exit from shell

Environment Status – פקודות סטטוס סביבה

פקודה	תיאור
ls [d] [f...]	List files in directory
ls -1 [f...]	List files in detail
alias [name]	Display command aliases
printenv [name]	Print environment values
quota	Display disk quota
date	Print date & time
who	List logged in users
whoami	Display current user
finger [username]	Output user information
chfn	Change finger information
pwd	Print working directory
history	Display recent commands
! n	Submit recent command n

File Manipulation - פקודות קבצים

פקודה	תיאור
vi [f]	Vi fullscreen editor
emacs [f]	Emacs fullscreen editor
ed [f]	Text editor
wc f	Line, word, & char count
cat f	List contents of file
more f	List file contents by screen
cat f1 f2 >f3	Concatenates f1 & f2 into f3
chmod mode f	Change protection mode of f
cmp f1 f2	Compare two files
cp f1 f2	Copy file f1 into f2
sort f	Alphabetically sort f
split [-n] f	Split f into n-line pieces
mv f1 f2	Rename file f1 as f2
rm f	Delete (remove) file f
grep 'ptn' f	Outputs lines that match ptn
diff f1 f2	Lists file differences
head f	Output beginning of f
tail f	Output end of f

20465
מהדורה פנימית
לא להפצה ולא למכירה
מק"ט 20465-5054