C Language

The Basics

By Nir Adar

מהדורה שניה - 18.4.2003
מהדורה ראשונה - 16.7.1998

שפת C - יסודות

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

אנא שלחו תיקונים והערות אל המחבר.

הקדמה למהדורה השניה

מסמך זה בא להציג לקורא את שפת C. המסמך מיועד למתחילים הבאים ללמוד את השפה.
מסמך זה איננו מתיימר להיות מדריך מלא ושלם של שפת C, אלא להציג מעט משפה נהדרת זו.
המהדורה הראשונה נכתבה ב-1998, ומאז קיבלתי הערות וביקורות רבות מגולשים שנעזרו במסמך.
מאז כתיבת המסמך ההוא העברתי מאות שיעורים בנושא שפת C, ובמהדורה חדשה זו אנסה להעביר לכם הקוראים את הדרך בה אני רואה את התכנות ב-C כיום.
במסמך המקורי הדגשתי את ההגדרות הפורמליות של השפה, ונתתי דגש מיוחד לתחבירי הפקודות ולהיבטים מתמטיים של השפה. לנושאים אלו חשיבות רבה, אולם במהדורה זו בחרתי דווקא לנסות להעביר לקורא ולהדגיש יותר את התמונה הכוללת - מהי שפת C, מהם הרכיבים שלה, ולמה רכיבים אלו הם חלק מהשפה.
כמו כן הושם דגש על דוגמאות מפורטות המראות נושאים שונים בשפה, וכן תרגילים אותם יוכל הקורא לתרגל בעצמו.

כולי תקווה שהמסמך יהיה לכם לעזר.

ניר אדר
אפריל 2003

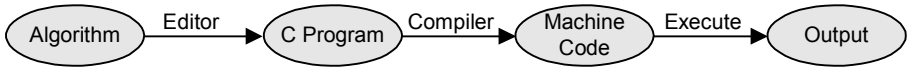
תוכן עניינים

מבוא ותוכנית ראשונה

מבוא

ככל שעוברים השנים, מחשבים נמצאים בשימוש רב יותר ויותר בחיינו - המחשב האישי חודר לכמעט כל בית, ומחשבים שונים משולבים במגוון יישומים - במכשירים סלולריים, במכוניות ואפילו בטלפונים סלולריים. מחשבים אלו מבצעים מגוון רחב של פעולות.
נתמקד רגע במחשב האישי - ה-PC. בתור משתמשים, אנו מפעילים תוכנות שונות על המחשב.
נניח שנרצה לבצע פעולה מסוימת על המחשב - למשל - כתיבת מסמך. סדר פעולות אפשרי בשבילנו הוא פתיחת תוכנת Word, הקלדת המסמך הרצוי ואז שמירתו לדיסק.
נשאל עכשיו את השאלה: כיצד התוכנה Word עובדת? כיצד תוכנות אחרות על המחשב עובדות? ויותר חשוב מכך - איך נוכל ליצור תוכניות חדשות שיתאימו למטרות ולצרכים שלנו?
התשובה: תוכנות נכתבות בעזרת שפות תכנות - שפות מוגדרות היטב המאפשרות לנו לומר למחשב מה ברצוננו שיבצע. במסמך זה נציג את אחת משפות התכנות הקיימות - שפת C.
אנשים החדשים לתחום התכנות מופתעים פעמים רבות כאשר מסתבר להם עד כמה המחשב הוא מכונה פשוטה, מבחינת הפקודות שאנו יכולים לומר לו לבצע. למשתמש הקצה נראה לעיתים שהמחשב הוא מעין כלי קסם המסוגל לבצע אינסוף פעולות, בעוד שלמעשה כל עושר התוכניות הקיימות מושג על ידי שפה נוקשה ובעלת מספר מילים מצומצם ביותר.
נסיר כעת מעט מהערפל ונציג כיצד אנו יכולים להפעיל על המחשב תוכניות שאנו כותבים.

בעזרת שפת תכנות אנו הופכים את הרעיונות שלנו לתוכנית שרצה במחשב.
אנו מספקים למחשב הוראות צעד אחד צעד כיצד עליו לבצע את המשימה המבוקשת והמחשב עוקב אחר הוראות אלו.
תהליך התכנות כולל לרוב תכנון מוקדם: כאשר אנו באים לכתוב תוכנית, ראשית נחשוב על סדרת הוראות למחשב, שביצוען יביא אל התוצאה הרצויה. סדרת הוראות אלו תכתב בשפה הקרובה מאוד לשפת בני אדם - בצורת אלגוריתם. האלגוריתם צריך להגדיר את הקלט אותו אמורה התוכנית לקבל, את העיבוד שהתוכנית אמורה לעשות על הקלט, ואת הפלט שהתוכנית תפיק בסיום.
לאחר שיש בידנו אלגוריתם, נעבור לממש אותו בשפת תכנות. נפתח עורך טקסט ונכתוב בו את התוכנית בשפת התכנות הנבחרת.
שפת C הינה שפה עילית - כלומר היא מכילה ביטויים הקרובים יחסית לשפה האנגלית, ומכילה מבני בקרה מורכבים. המחשב אינו מבין הוראות הכתובות בשפת C. מחשב מבין רק הוראות הכתובות בשפה הנקראת שפת מכונה. לפיכך, השלב הבא ביצירת התוכנית שלנו הוא תרגום התוכנית לשפת מכונה.
תוכנית הממירה את ההוראות שכתבנו בשפת תכנות להוראות בשפת מכונה אותן המחשב יודע לבצע, נקראת מהדר או Compiler.
לאחר שהמהדר הידר את התוכנית שכתבנו, אנו משתמשים בתוכנית נוספת הנקראת Linker שהופכת את התוכנית לקובץ EXE שניתן להריצו.
לכאורה תהליך זה ארוך ומסורבל. בפועל כלי העבודה המצויים כיום מאפשרים לבצע את תהליך יצירת התוכנה בקלות. שתי סביבות עבודה טובות הינן Borland C++ ו-Microsoft Visual C++.
המלצתנו למתכנת המתחיל היא לעבוד עם Borland C++, הנוחה יותר לעבודה עם תוכניות קטנות.
Microsoft Visual C++ נוחה יותר עבור פרויקטים גדולים.
סביבות עבודה אלו כוללות בתוכן עורך, מהדר, מקשר (Linker) ועוד, והן מאפשרות לבצע תהליך זה במהירות על ידי תפריטים.



תוכנית ראשונה ב-C

נציג כעת תוכנית ראשונה בשפת C. התוכנית מדפיסה את המילים "Hello, World" על המסך.

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

2)    int main()
3)    {
4)         printf("Hello, World!\n");
5)         return 0;
6)    }
```

הערה: מספור השורות איננו חלק מהתוכנית, אלא נועד לצורך התייחסות והסברים.

נסביר את תפקיד כל אחת משורות התוכנית:

1. הוראה למהדר להכניס לפני התוכנית את קובץ הכותרת (Header) בשם stdio.h. קובץ זה מכיל הגדרות של פקודות שונות ב-C, בין היתר של הפקודה printf() בה אנו משתמשים בתוכנית זו. קובץ זה מכיל הרבה מפקודות הקלט/פלט של שפת C.
2. הגדרת פונקציה בשם main. פונקציה ב-C היא קטע של הוראות שניתן להריצו, ושיכול להחזיר ערך בסיום, המסמל משהו. ב-C תמיד הפונקציה main רצה כאשר התוכנית מתחילה.
כרגע די לנו בעובדה שריצת התוכנית מתחילה בבלוק זה המוגדר בעזרת main(). אנו עוד נעסוק בנושא הפונקציות בשפת C בהרחבה.
3. הסימן { מציין את התחלת הבלוק בו יהיו ההוראות של main. הסימן } מציין את סוף בלוק הפקודות של הפונקציה main. ככלל, כל בלוק בשפת C מתחיל בסימן { ומסתיים ב-}.
4. printf היא פונקציה המשמשת להדפסת המחרוזת המצויה בין המרכאות. (מחרוזת - רצף אותיות). הסימן \n הוא סימן מיוחד בשפת C, שמשמעותו מעבר לשורה הבאה.
5. return 0 אומר לתוכנית להחזיר למערכת ההפעלה 0, כלומר לסיים את התוכנית ולהודיע למערכת ההפעלה שהתוכנית הסתיימה בלי שגיאה (מקובל ש-0 מייצג סיום מוצלח של התוכנית, וכל מספר אחר מייצג שגיאה).

כאשר נריץ תוכנית זו, הטקסט "Hello, World" יודפס על המסך.

הערה: נשים לב שבתוך הפונקציה main(), כל אחד מהשורות הסתיימה ב- ; (נקודה פסיק).
בשפת C, למעט עבור פקודות מסויימות מאוד אותן נפרט, כל ביטוי מסתיים בנקודה פסיק.
אחת השגיאות הנפוצות של מתכנתים מתחילים היא לשכוח לשים ; במקומות הדרושים, וחשוב לשים לב לכך.

מינוח: נהוג לכנות את הפקודות בשפת C המרכיבות את התוכנית שלנו בשם קוד התוכנית.

תרגיל

כתוב תוכנית שתדפיס את הטקסט "I Love C" על המסך, כשהוא מוקף במלבן הבנוי מכוכביות (*).

פתרון

מטרתו של תרגיל זה היא לגרום לקורא לנסות ולכתוב תוכנית ראשונה בעצמו, להתנסות בתחביר הבסיסי של C, ולראות כיצד מהדרים תוכנית וכיצד מריצים אותה.

פתרון התרגיל:

```
#include <stdio.h>

int main()
{
     printf("************\n");
     printf("* I love C *\n");
     printf("************\n");

     return 0;
}
```

זיכרון המחשב ומשתנים

זיכרון המחשב

כאשר נכתוב תוכנית, נרצה לאפשר לה לשמור מידע, לנתח אותו ולבצע עליו פעולות.
את מידע התוכנית נשמור בזיכרון המחשב.

נתאר קצת את זיכרון המחשב:
היחידה הבסיסית ששומרת במחשב מידע מכונת ביט (Bit). ביט היא יחידה היכולה לקבל רק שני ערכים: 0 או 1, דלוק או מכובה. קבוצה של 8 ביטים נקראת בית (Byte). (נעיר כי במחשבים מודרניים בית הוא לרוב יותר מ-8 ביטים, אולם לצורך ההסבר נניח שבית הוא בן 8 ביטים).

הזיכרון של המחשב בנוי מרצף של בתים, המכילים בתוכם נתונים. הבתים מכונים גם תאים.
ניתן לדמיין את הזיכרון של המחשב בתור רצף של בתים, שלכל אחד מהם יש מספר מזהה ייחודי, ובכל אחד מהם נשמרים נתונים.
למספר המזהה הייחודי של התא נקרא כתובת.
נביט למשל בשרטוט הבא.

1001	1002	1003	1004	1005	1006	1007
01001110	10101010	00101110	10101011	10000111	11100111	10101101

שרטוט זה הינו דוגמא איך יכול להראות קטע מהזיכרון.
המספרים 1001-1007 מייצגים את כתובת התאים. האפסים והאחדות מייצגים נתונים שיכולים להיות בתאים.

כיצד נשמור בזיכרון נתונים אחרים פרט לאפסים ואחדות, למשל - מספרים, אותיות וכו'?
התשובה: בעזרת ייצוג של מספרים ואותיות כרצפים מוסכמים של אפסים ואחדות, ופיענוח של האפסים והאחדות על ידי המחשב.
בלי להכנס למתמטיקה שמאחורי הנושא, נאמר כי קיימת שיטה לקשר בין המספרים המוכרים לנו, הנקראים מספרים בבסיס עשרוני, לרצפים של אפסים ואחדות, המכונים מספרים בינריים, באופן חד חד ערכי.
כמו כן, אם נקבע כי לכל אות בשפה קיים מספר מייצג, אז נוכל לשמור גם את האותיות בזיכרון כמספרים.

שפת C מפשטת לנו את העבודה עם הזיכרון על ידי הפשטה. בשפת C קיימים מספרים ואותיות. אנו יכולים להשתמש בהם בתוכניות שלנו, והמהדר של C הוא זה שימיר אותם בשבילנו לאפסים ואחדות שישמרו בזיכרון המחשב.

משתנים

על מנת שהתוכנית תוכל לשמור מידע, אחד מהאלמנטים של שפת C הוא משתנים.
משתנה הוא למעשה תא בזיכרון, לו אנו נותנים שם מיוחד שיזהה אותו, ובו אנו יכולים לשמור מידע.
בשפת C ישנם סוגים שונים של משתנים: משתנים שמיועדים לשמור מספרים שלמים, משתנים המיועדים לשמור תווים (אותיות), ומשתנים המיועדים לשמור מספרים רציונליים.
ההגדרה של משתנה בשפת C מורכבת משני חלקים: סוג המשתנה אותו אנו רוצים ליצור, והשם שאנו רוצים להגדיר עבור המשתנה.
מבחינה תחבירית, ההצהרה על משתנה נראית כך:

```
<Variable type> <Variable name>;
```

סוגי המשתנים העיקריים של שפת C:
א. משתנים המכילים מספרים שלמים (int).
ב. משתנים המכילים אותיות (char).
ג. משתנים המכילים מספרים רציונליים (double).

לדוגמא:

```
int x;
```

בהצהרה זו הגדרנו משתנה מסוג int, שזהו מספר שלם, וקראנו לו x.

ניתן להציב לתוך המשתנה ערכים, כך :

```
x = 4;
```

אנו מכניסים לתוך התא בזיכרון שמכונה בתוכנית שלנו x, את המספר 4.
ניתן להכריז על יותר ממשתנה אחד בו זמנית, כך:

```
int x, y;
```

הגדרנו 2 משתנים, x ו-y, שניהם מסוג מספרים שלמים.

על מנת לשמור תו כלשהו בתוך משתנה מסוג char נשתמש בגרשיים בודדות, ובתוכן נשים את התו הרצוי לנו.
למשל:

```
char ch = 'A';
```

יצרנו משתנה בשם ch ושמנו בתוכו את האות A.

נשים לב כי התו 'A' שונה מהמשתנה A. הגרשיים מציינות שמדובר בתו, ואילו ללא הגרשיים היה זה שם משתנה או קבוע.

נעצור לרגע את המשך ההסבר על המשתנים ונראה כיצד אנו משתמשים ב-C לפעולות קלט/פלט.
לאחר מכן נמשיך בהצגת והרחבת נושא המשתנים.

פונקציות קלט/פלט

בעזרת פונקציות קלט ופלט התוכניות שנכתוב יתקשרו עם העולם.
נציג פונקציות המשמשות להדפסת נתונים על המסך, וכן פונקציות המשמשות לקלט מהמקלדת.
על מנת להשתמש בתוכנית שלנו בפונקציות קלט/פלט, אנו צריכים לכלול את הקובץ stdio.h בתוכנית, בעזרת הפקודה:

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

stdio זהו קיצור של standard input/output. זוהי ספרית פקודות של שפת C, המכילה את פקודות הקלט פלט השימושיות ביותר של השפה.

הפונקציות printf/scanf

printf היא פונקציה המדפיסה דברים על המסך.
ראינו כבר את הצורה הבאה של השימוש בפונקציה printf:

```
printf("מחרוזת");
```

כאשר השתמשנו ב-printf בצורה כזו, המחרוזת שהקלדנו הודפסה על המסך.

באופן כללי, המבנה של הפונקציה printf הוא

```
printf("מחרוזת בקרה", arg1, arg, ..., argX);
```

מחרוזת הבקרה היא מחרוזת שאנו רוצים שתודפס על המסך, המכילה בתוכה גם תווים מיוחדים, שיפוענחו על ידי printf ויוחלפו לתווים אחרים. לאחר מחרוזת הבקרה תופיע רשימת משתנים/ביטויים, שתהיה קשורה אל מחרוזת הבקרה. כל ביטוי יתאים לאחר מסימני הבקרה שיהיו במחרוזת הבקרה.
לדוגמא:

```
printf("hello %d\n", 2);
```

%d הוא סימן בקרה המציין מספר שלם.
הפלט על המסך שיווצר כאשר הפקודה הבאה תתבצע הינו:

```
hello 2
```

כאמור, נוכל גם להדפיס משתנים על המסך:

```
int x = 10;
printf("%d\n", x);
```

על המסך יודפס המסר 10.

על מנת להדפיס משתנים מסוג double נשתמש בסימן הבקרה %lf, ועבור משתנים מסוג char נשתמש ב-%c, למשל:

```
double d = 10.7;
printf("%lf", d);
```

נביט בשתי השורות הבאות:

```
printf("from sea");
printf(" to C");
```

התוצאה תהיה שורה אחת של פלט. אם לא נשים את הסימן \n בסוף השורה, איננו עוברים לשורה הבאה באופן אוטומטי.
scanf היא פונקציה המאפשרת לקלוט מהמשתמש, מהמקלדת, ערכים לתוך משתנים.
המבנה הכללי של הפונקציה הוא:

```
scanf("מחרוזת בקרה", &a, &b, &..., &x);
```

כאשר a, b, ... הינם משתנים לתוכם אנו רוצים לקרוא נתונים, ומחרוזת הבקרה היא רצף סימני בקרה, שאומרים מהו הסוג של כל אחד מהמשתנים שיקראו.

לפני שם המשתנה לתוכו אנו רוצים לקלוט את המידע אנו שמים את האופרטור &, שמשמעותו "הכתובת של". אנו רוצים שהנתונים שנקראים מהמקלדת יכנסו אל כתובת של המשתנה, כלומר אל תוך המשתנה, ולכן אנו שמים אופרטור זה לפני כל משתנה.
נעיר כי טעות נפוצה היא לשכוח להשתמש באופרטור זה כאשר משתמשים בפונקציה scanf.
דוגמא לשימוש בפונקציה:

```
int i;
scanf("%d", &i);
```

נדגים שימוש בשתי פונקציות אלו יחד:

```
#include <stdio.h>

int main(void)
{
      int i;
      printf("Please put a number: ");
      scanf("%d", &i);
      printf("You entered %d!\n", i);
      return 0;
}
```

הפלט של התוכנית:

```
Please put a number :
```

התוכנית תחכה שנקליד מספר. נכתוב למשל את המספר 10 ונלחץ Enter. על המסך יופיע:

```
You entered 10!
```

והסמן יעבור שורה. לאחר מכן התוכנית תסתיים.

עיצוב הטקסט בעזרת printf

על מנת להציג נתונים באופן מסודר על המסך, printf נותנת לנו כלים לעצב את הפלט שלה, על ידי תוספות אופציונליות אל סימני הבקרה. נביט בתוכנית הבאה:

```
#include <stdio.h>
int main()
{
	double d;
	printf("Please enter a number: ");
	scanf("%lf", &d);
	printf("The number is %.2lf\n", d);
	return 0;
}
```

התוכנית תדפיס את המספר שהמשתמש מקליד, בדיוק של שתי ספרות לאחר הנקודה.
למשל, נניח כי המשתמש הקליד את המספר 62.54453, על המסך יודפס 62.54 בלבד.
השליטה על כמות הספרות שיוצגו נעשית על ידי הוספת הנקודה אחרי ה-%, ולאחר מכן ציון מספר הספרות המבוקש.

אמצעי עיצוב נוסף הוא קביעת רוחב מינימלי לפלט, עימוד הפלט, למשל, נביט בתוכנית הבאה:

```
#include <stdio.h>
int main()
{
	printf("%6d%6d%6d\n", 1, 20, 300);
	return 0;
}
```

על ידי כתיבת מספר מיד לאחר ה-%, אנו קובעים את אורך הפלט המינימלי לערכו של מספר זה.
התוצאה תהיה: 5 רווחים, לאחריהם המספר 1. לאחר מכן יהיו עוד 4 רווחים, ולאחריהם המספר 20, ולסיום יהיו שלושה רווחים, ואחריהם המספר 300.

```
     1    20   300
```

היישור שקיבלנו הוא יישור לימין. בעזרת הסימן - (מינוס), נוכל לקבוע יישור לשמאל. לדוגמא:

```
#include <stdio.h>
int main()
{
      printf("%-6d%-6d%-6d\n", 1, 20, 300);
      return 0;
}
```

כעת הפלט יהיה:
1 ולאחריו 5 רווחים, 20 ולאחריו 4 רווחים, 300 ולאחריו 3 רווחים.

```
1     20    300
```

ניתן לשלב בין שני העיצובים שראינו:

```
#include <stdio.h>
int main()
{
      double d;
      printf("Please enter a number: ");
      scanf("%lf", &d);
      printf("The number is %6.2lf\n", d);
      return 0;
}
```

כעת המספר יודפס בדיוק של שתי ספרות אחרי הנקודה, והוא ירופד ברווחים במידה ואורכו קטן מ-6.

הפונקציות getchar/putchar

הפונקציה getchar משמשת לקליטת תו בודד מהמשתמש. לדוגמא:

```
char ch;
ch = getchar( );
```

הפוקנציה putchar משמשת להדפסת תו בודד על המסך, לדוגמא:

```
putchar('g');
```

ניתן להשתמש תמיד ב-printf/scanf ולא בפונקציות אלו, אולם אם נרצה להדפיס רק תו בודד, נעדיף לעיתים להשתמש ב-putchar/getchar.

משתנים

סוגי משתנים

המשתנים השונים הקיימים בשפת C הינם:

char	signed char	unsigned char
short	int	long
unsigned short	unsigned int	unsigned long
float	double	long double

במסמך זה נעסוק רק בחלק מסוגי המשתנים הקיימים בשפה. נציג כעת שוב את המשתנים, ונרחיב כעת על כל אחד מהם. נציין כי ניתן להפעיל על כל המשתנים את פעולות החשבון הבסיסיות: /,*,-,+.

מספרים שלמים (int)

משתנים שהם מסוג int הם משתנים המכילים מספרים שלמים בלבד. כמו כן, תוצאת פעולה אריתמטית בין משתנים מסוג int היא מספר שלם בהכרח. נביט בקוד הבא:

```
#include <stdio.h>

int main()
{
	int i = 5, j = 5.7;
	printf("%d %d\n", i, j);
	return 0;
}
```

נשאל: מה תדפיס הפונקציה printf? התשובה: הפונקציה תדפיס 5 5.
גם הערך של i וגם הערך של j שווה ל-5, למרות שלכאורה השמנו לתוך j מספר רציונלי, וזאת מכיוון שמשתנים מסוג int מכילים רק מספרים שלמים.

הקוד הבא מדגים כיצד אנו קולטים מידע מהמשתמש ומשתמשים בו. נקלוט שני מספרים שלמים מהמשתמש, נחשב את סכומם ונדפיס על המסך את התוצאה:

```
#include <stdio.h>

int main()
{
	int i, j, sum;
	printf("Please enter two numbers: ");
	scanf("%d%d", &i, &j);
	sum = i + j;
	printf("%d + %d = %d\n", i, j, sum);
	return 0;
}
```

משתנים רציונליים (double)

משתנים מסוג double מחזיקים מספרים ממשיים.
נציג משתנה זה בעזרת תוכנית קצרה, הקולטת שלושה מספרים מהמשתמש, ומחשבת את הממוצע שלהם.

```
#include <stdio.h>

int main()
{
	double d1, d2, d2, avg;
	printf("Please enter 3 numbers: ");
	scanf("%lf%lf%lf", &d1, &d2, &d3);
	avg = ( d1 + d2 + d3 ) / 3;
	printf("The average is %lf\n", avg);

	return 0;
}
```

תווים (char)

char הוא משתנה שלרוב נשמור בו תווים. תו הוא כל אות, מספר, או סימן אחר שאנו מסוגלים להציג על המסך.
כפי שכבר הזכרנו, כל תו מיוצג במחשב על ידי מספר, ולכן ניתן גם להתייחס למשתנים מסוג char כמו אל משתנים המכילים מספרים שלמים קטנים.
המספרים המתאימות לאותיות שאנו שומרים במשתנים מסוג char נשמרים לפי סטנדרט בשם ASCII, המתאים לכל אות מספר המתאים לה.
ההתיחסות למשתנה מסוג char - בתור אות או בתור מספר, תלויה בנו.
נציין כהערה כי ערך ה-ASCII של האות 'a' הוא 97, ונביט בדוגמא הבאה:

```
#include <stdio.h>

int main()
{
	char c = 'a';
	printf("%d %c\n", c, c);
	return 0;
}
```

על המסך יודפס 97 ולאחריו תודפס האות a.
הסיבה: במשתנה c נשמר למעשה המספר 97. כאשר קראנו ל-printf עם סימן הבקרה %d אמרנו לה להתייחס אל המשתנה כאל מספר, וכאשר השתמשנו ב-%c אמרנו לה להתייחס אליו כאל אות, ואז המספר 97 למעשה ייצג את ערך ה-ASCII של האות שאנו רוצים להדפיס.

בשפת C ישנם מספר תווים מיוחדים. את חלקם כבר ראינו, למשל התו '\n' שמשמעותו מעבר לשורה חדשה. בדומה לתווים האחרים, לכל תו כזה יש ערך מספרי, ובנוסף ישנה דרך מיוחדת לכתוב אותו ב-C.

נציג את התווים המיוחדים בטבלה:

Name of character	Write in C	int value
alret	\a	7
backslesh	\b	92
backspace	\r	8
carriage return	\î	13
double quote	\î	34
form feed	\f	12
horizontal tab	\t	9
new line	\n	10
null character	\0	0
single quate	\í	39
vertical tab	\v	11

מספרים שלמים גדולים (long)

טווח הייצוג של משתנים מסוג int הינו מוגבל. הטווח אינו מוגדר באופן חד ערכי על ידי השפה. טווח המספרים שאנו יכולים לשמור במשתנה מסוג int תלוי במהדר בו אנו משתמשים.
עם זאת, ב-C מוגדר סוג משתנה נוסף עבור משתנים שלמים, והוא long.
long כאמור הוא משתנה השומר משתנים שלמים, ובנוסף לכך C מבטיחה לנו שגודלו יהיה גדול או לכל הפחות זהה לגודלו של משתנה מסוג int.

<u>המרות</u>

על מנת להציג את נושא ההמרות, נביט בתוכנית הבאה, ונשאל את עצמנו מה יהיה הפלט שלה.

```
#include <stdio.h>

int main()
{
	int i = 12 / 5;
	double d = 12 / 5;
	printf("%d %lf\n", i, d);

	return 0;
}
```

הפלט של התוכנית יהיה: 2 2.000
נבין מדוע זהו פלט התוכנית.
עבור המשתנה i: i הוא משתנה מסוג שלם, ולכן כאשר ביצענו חלוקה, ייכנס תמיד רק החלק השלם של התוצאה לתוך המשתנה, ולכן i מכיל 2.
עבור המשתנה d: d הוא משתנה מסוג double, ולכן לכאורה כאשר ביצענו חלוקה, היינו אמורים לקבל בתוך המשתנה d את המספר 2.4, אולם קיבלנו בתוכו את המספר 2.000.

הסיבה לכך: 12 ו-5 הם מספרים שלמים. שפת C ביצעה את החלוקה של 12 ב 5 כמספרים שלמים, כלומר שמרה רק את החלק השלם, 2, ואז המירה את התוצאה למשתנה מסוג double ושמרה אותו במשתנה d.
C מגדירה חוקים ברורים היטב מה יהיה סוג התוצאה של פעולות בין שני משתנים מסוגים שונים, או מאותו סוג. בשפת C המשתנים מתחלקים לשתי קבוצות: מספרים שלמים ומספרים לא שלמים.
בקבוצת המספרים השלמים, ניתן למצוא את הטיפוסים unsigned long, unsigned, long ו-int.
בקבוצת המספרים הלא שלמים, ניתן למצוא את הטיפוסים long double, double ו-float.
כאשר אנו מבצעים פעולות המערבות מספרים שלמים בלבד, התוצאה תהיה מספר שלם.
כאשר אנו מבצעים פעולות המערבות מספרים עשרוניים בלבד, התוצאה היא מספר עשרוני.
בפעולות מעורבות, התוצאה תהיה מספר עשרוני.
אם הפעולה מערבבת משתנים מגדלים שונים, התוצאה תהיה מסוג המשתנה הגדול יותר. למשל, תוצאת פעולה בין משתנה מסוג int למשתנה מסוג long תהיה מסוג long.

נביא כעת טבלה שתרכז מספר דוגמאות לביטוים ולסוג המוחזר מהביטוי.
קיצורים בהם נשתמש בדוגמא:

c=char	s=short	i=int	u=unsigned	l=long

ביטוי	סוג מוחזר
c-s / i	int
u * 2.0 - i	double
l+3	int
l+3.0	double
d+s	double
2 * l	long
u * 7 - i	unsigned
f * 7 - i	float
7 * s * ul	unsigned long
u - ul	unsigned long
u-l	אין תשובה חד משמעית - תלוי במחשב

חמושים בידע חדש זה, נרצה לשנות את התוכנית הקודמת, כך שתשמור את המספר בתוך המשתנה d כראוי. השינוי הבא לתוכנית יבצע את הנדרש:

```
#include <stdio.h>

int main()
{
	int i = 12 / 5;
	double d = 12.0 / 5;
	printf("%d %lf\n", i, d);

	return 0;
}
```

כעת, מכיוון שאחד מהמספרים שחישובם נותן לנו את d איננו מספר שלם, התוצאה גם היא איננה מספר שלם, ולכן לא תיחתך, ובמשתנה d יישמר הערך 2.4.

לפעמים נרצה להמיר באופן מפורש סוג משתנה אחד לסוג משתנה אחר, כבר במהלך החישוב. על מנת לעשות זאת נשים סוגרים עם הטיפוס שאליו רוצים להמיר לפני הביטוי שאותו אנו רוצים להמיר, בצורה כזו:

ביטוי (טיפוס)

למשל, ערכו של הביטוי (int)7.2 הינו 7.

המרה בדרך זו נקראת casting.

נכתוב שוב את התוכנית, והפעם נפתור את חוסר נכונות התוכן של המשתנה d בעזרת casting.

```
#include <stdio.h>

int main()
{
	int i = 12 / 5;
	double d = (double)12 / 5;
	printf("%d %lf\n", i, d);

	return 0;
}
```

כעת התוכנית תוציא פלט כמצופה.

תרגילים

תרגיל 1

כתוב תוכנית המקבלת שני מספרים שלמים ומדפיסה את החיבור ביניהם, את החיסור ביניהם ואת מכפלתם.

תרגיל 2

מה תדפיס התוכנית הבאה:

```
#include <stdio.h>

int main()
{
	int i;
	double d;
	i = d = 3.2;
	printf("%d %lf\n", i, d);
	d = i = 3.2;
	printf("%d %lf\n", i, d);
	return 0;
}
```

תרגיל 3

כתוב תוכנית המקבלת שלושה מספרים שלמים מן המשתמש ומדפיסה את הממוצע שלהם.

פתרונות

פתרון 1

מטרת התרגיל: תרגול עבודה עם משתנים.
פתרון אפשרי:

```
#include <stdio.h>

int main()
{
	int x, y;
	printf("Please enter 2 numbers: ");
	scanf("%d%d", &x, &y);
	printf("Sum = %d\nSub = %d\nMul = %d\n", x+y, x-y, x*y);
	return 0;
}
```

טעויות נפוצות בפתרון תרגיל זה:

- אי השמת נקודה פסיק בסוף כל ביטוי.
- התעלמות מהצורך להשתמש באופרטור & בפונקציה scanf.

פתרון 2

מטרת התרגיל הינה להדגיש את המרות הסוגים ש-C מבצעת.

בביטוי הבא:

```
i = d = 3.2;
```

יוצב ב-d 3.2, וב-i יוצב 3. לעומת זאת, בביטוי הבא:

```
d = i = 3.2;
```

המספר 3.2 יהפוך לשלם 3 על מנת שיוצב ב-i, ואילו ב-d יוצב 3.0.

פתרון 3

הדגש שהיה צריך לשים לב אליו בתרגיל זה הוא שלמרות שאנו קולטים שלושה מספרים שלמים, הממוצע איננו בהכרח מספר שלם. לכן, כאשר אנו מחשבים את הממוצע עלינו לבצע casting למשתנים ל-double, וכן, אם בפתרון שלנו נרצה גם לשמור את הממוצע נדרוש שהמשתנה שישמור את הממוצע יהיה מסוג double.

פתרון אפשרי לשאלה:

```
#include <stdio.h>

int main()
{
	int i1, i2, i2;
	double avg;
	printf("Please enter 3 numbers: ");
	scanf("%d%d%d", &i1, &i2, &i3);
	avg = ( (double)i1 + i2 + i3 ) / 3;
	printf("The average is %lf\n", avg);
	return 0;
}
```

מרכיבי שפת C

סקירה קצרה של מרכיבי השפה

מילות מפתח (Key words)

מילים השייכות לשפת C, והן למעשה אבני הבניין הבסיסיות של השפה. סוגי המשתנים השונים, למשל int, long, הינם מילים שמורות. גם פקודות עבור לולאות והסתעפויות אותן נציג בהמשך הן מילים שמורות. דוגמאות נוספות למילים שמורות: auto, break, case, car, const, do, double, else, enum, for, int, long והרשימה עוד ארוכה.

מזהים (identifiers)

מזהים הם רצף של מספרים ואותיות המקיימים את הכלל הבא: האות הראשונה שלהם היא אות אנגלית או _ (קו תחתון), וכל האותיות הבאות הינן אותיות, מספרים או קו תחתון.
מזהים ב-C משמשים כשמות של משתנים, שמות של פונקציות ושמות של קבועים.

קבועים (Constants)

ישנם שני סוגי קבועים בשפת C- קבועי מספרים וקבועי מחרוזת
קבועי מספרים : 6, 4, 7.5, 12.39, -27, 666, 'a'
קבועי מחרוזת : "A", "%d", "blah", "TIME_TO_GO"

הערות

הערות ב-C הן קטעים הנמצאים בין /* */. המהדר מתעלם לחלוטין מתוכן הערות, ולכן נוכל לכתוב בהערות באנגלית למשל. נהוג להשתמש בהערות של C על מנת להוסיף הסברים כיצד התוכנית עובדת ומה היא עושה בכל שלב ושלב, על מנת שמתכנתים עתידיים שייגשו אל הקוד שכתבנו יבינו אותו יותר בקלות.

אופרטורים (Operators)

אופרטור בשפת C: +, -, *, =, / ,=+, <, % ,-, ++ ועוד.
ניתן לחלק את האופרטורים ב-C למספר קבוצות:
1. אופרטורים פשוטים של מתמטיקה : (), +, *, /, -
2. אופרטורים של השמה : =, =+, = -, =*
3. אופרטורים של השוואה : ==, =>, =<, <, >
4. אופרטורים לוגיים: ||, &&, !
5. ++, --

ביטויים

צירוף של מספרים ואותיות שביניהם יש אופרטור. לכל ביטוי ב-C יש ערך שניתן להשתמש בו.
דוגמא לביטוי: a+b, כאשר a,b משתנים שהוגדרו קודם לכן בתוכנית. דוגמא:
נגדיר את המשתנים הבאים:

```
int a=1, b=2, c=3, d=4;
```

נביט בביטויים אפשריים ובערך שלהם. נשים לב שלביטוי יכול להיות יותר מייצוג יחיד.

expression	equal expression	value
a*b/c	(a*b)/c	0
a*b%c+1	((a*b)%c)+1	3
++a*b-c--	((++a)*b)-(c--)	1
7- -b*++d	7-((-b)*(++d))	12

אופרטורים

ניתן כעת דגש לאופרטורים של שפת C. לכאורה נסטה כעת אל קטע מעט מתמטי ו-"יבש", אך הבנה מעמיקה של האופרטורים של שפת C חיונית על מנת שנוכל לנצל את השפה.

אופרטורים של השמה

עד כה ראינו אופרטור השמה אחד בלבד והוא האופטור =.
שפת C מאפשרת לנו אופרטורים נוספים של השמה, לצורך קיצור ונוחות.
נראה למשל את האופרטור =+. נביט בקוד הבא:

```
x += 7;
```

המשמעות של הפקודה: הוסף לתוכן של x את המספר 7. השורה הבאה זהה בפעולתה לקודמת:

```
x = x + 7;
```

באופן דומה, האופרטור =- משמעותו לחסר מהמשתנה את הביטוי שבצד ימין של האופרטור.

האופרטורים ++/--

אופרטורים אלו פועלים על משתנה בודד. האופרטור ++ מגדיל את ערכו של המשתנה ב-1, ואילו האופרטור -- מקטין את ערכו של המשתנה ב-1.
אם האופרטור ייכתב משמאל למשתנה עלו הוא פועל, הערך של המשתנה ישתנה לפני ביצוע הפקודה הנוכחת. אם האופרטור ייכתב מימין, ערך המשתנה ישתנה לאחר ביצוע הפקודה הנוכחית.
לדוגמא:

```
int a=2, b=2, c, d;
c = a++;
d = ++b;
```

קטע הקוד לעיל זהה לקטע הקוד הבא מבחינת פעולתו:

```
int a=2, b=2, c, d;
c = a;
a = a + 1;
b = b + 1;
d = b;
```

<u>אופרטורים של השוואה</u>

כאמור, לכל ביטוי בשפת C יש ערך. ניתן להתייחס אל כל ביטוי כאל ביטוי לוגי, כאשר מוגדר:
אם ערך הביטוי הוא 0, אזי הערך הלוגי המתאים לו הוא שקר (FALSE).
עבור כל ביטוי השונה מ-0, הערך הלוגי המתאים לו הוא אמת (TRUE).
למשל, ערכו של הביטוי 5 הוא אמת. אם נגדיר משתנה מסוג int בשם x, ונשים בו 0, אזי ערכו של הביטוי x הוא שקר.
שפת C מכילה מגוון של אופרטורים של השוואה. ניתן ליצור ביטויים לוגיים, הכוללים אופרטורים אלו.
למשל: ערכו של הביטוי (2<3) הוא אמת (שווה ל-1), בעוד שערכו של הביטוי (2>=3) הוא שקר (שווה ל-0).

לדוגמא, נביט בקטע הקוד הבא:

```
int x = 3 > 2;
printf("%d\n", x);
```

על המסך יודפס המספר 1.
נשים לב כי למרות שאמרנו כי כל מספר שונה מ-0 הוא אמת, כאשר בשפה אנו משתמשים בביטוי לוגי, ערכו 1 אם הוא אמת ו-0 אם הוא שקר.

אופרטורים של השוואה בשפת C:

אופרטור	משמעות
<	קטן
>	גדול
<=	קטן שווה
>=	גדול שווה
==	שווה
!=	לא שווה

טעות נפוצה היא להתבלבל בין אופרטור ההשמה = לאופרטור ההשוואה ==. נביט בתוכנית הבאה:

```
#include <stdio.h>

int main()
{
	int x1, y1 = 7, z1 = 8;
	int x2, y2 = 7, z2 = 8;

	x1 = y1 = z1;
	x2 = y2 == z2;

	printf("%d %d\n", x1, x2);

	return 0;
}
```

מה תדפיס התוכנית? התוכנית תדפיס את המספר 8, ולאחריו את המספר 0.
בשורה הצבועה בכחול, התבצעו הפעולות הבאות: תחילה הוצב הערך 8 בתוך המשתנה y1, ולאחר מכן ערך זה הושם ב-x1.
לעומת זאת, בשורה הצבועה בירוק, התבצעה השוואה בין הערך של y2 לערך של z2. מכיוון שערכם שונה, ערך הביטוי y2 == z2 שווה ל-"y2 - z2" - הוא שקר, ולכן ב-x2 הוצב הערך 0.

<u>אופרטורים לוגיים</u>

אופרטור	משמעות
!	NOT (אופרטור אונרי)
&&	AND לוגי (אופרטור בינארי)
||	OR לוגי (אופרטור בינארי)

אופרטור אונרי הוא אופרטור שפועל על ביטוי אחד. אופרטור בינארי הוא אופרטור שפועל בין שני ביטויים.

כאמור, בשפת C כל תוצאה שהיא שונה מ-0 מתפרשת כאמת ו-0 הוא שקר.
בטבלה הבאה, 1 מציין בטבלה זו כל אחד השונה מ-0.

!x	x
1	0
0	1

דוגמאות:

ערך	ביטוי
0	!0.001
1	!!'A'
0	!('Z' > 'A')

דוגמאות לשימוש באופרטורים הבינאריים:

x	y	x && y	x || y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

האופרטור sizeof

אופרטור זה מחזיר את מספר הבתים של הביטוי, טיפוס או משתנה הנמצא מימינו.
אופן השימוש באופרטור:

```
sizeof(expression)
sizeof(type)
```

לדוגמא:

```
#include <stdio.h>

int main()
{
      double j = 5.5;
      printf("In my computer: \n");
      printf("The size of char variable is %d\n",
                  sizeof(char));
      printf("The size of int variable is %d\n", sizeof(int));
      printf("The size of j is %d\n", sizeof(j));

      return 0;
}
```

הפלט של תוכנית זו משתנה ממחשב למחשב, מכיוון שלא מוגדר ב-C גודל קבוע של בתים עבור כל סוג משתנה. פלט אפשרי של התוכנית יכול להיות:

```
In my computer:
The size of char variable is 1
The size of int variable is 2
The size of j is 8
```

הערך המוחזר על ידי האופרטור sizeof שונה בין מערכת הפעלה אחת לשניה ובין קומפליר לקומפילר.
עם זאת, שפת C מבטיחה לנו את קיום התנאים הבאים:

sizeof(short) ≤ sizeof(int) ≤ sizeof(long)
sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)
sizeof(signed) ≤ sizeof(unsigned) ≤ sizeof(int)

אופרטור הפסיק

ניתן לכתוב מספר ביטויים אחד אחרי השני, המופרדים ביניהם על ידי פסיקים.
הערך של הביטויים שווה לביטוי אחד. הערך שיוחזר הוא הערך של הביטוי האחרון.

דוגמא 1:

```
x = 1, y = 7 + 3, ++z;
```

דוגמא 2:

```
#include <stdio.h>

int main()
{
        int x, y, z;
        x = (y = 4, z = 5);
        printf("%d\n", x);
        return 0;
}
```

על המסך יודפס המספר 5.
יש לציין כי השימוש באופרטור זה מאוד לא נפוץ.

הסתעפויות

עד כה, התוכניות שהצגנו יכלו לקלוט קלט מהמשתמש, לבצע עליו כמה עיבודים, ולהציג את התוצאה על המסך.
נרצה להוסיף תחכום לתוכניות שלנו, וזאת על ידי כך שניתן את האפשרות לתוכנית להגיב על הקלט, ולפעול בצורה שונה עבור קלטים שונים.
על מנת להשיג פונקציונאליות זו, נשתמש במילה השמורה if - המיועדת להסתעפויות מותנות.

שימוש בסיסי בפקודה יעשה בצורה הבאה:

```
if (ביטוי) פקודה;
```

אם הערך של הביטוי הוא אמת, כלומר שונה מ-0, אז הפקודה שלאחר הביטוי מתבצעת. לדוגמא:

```
if (2 < 3) printf("hello!\n");
```

אם נרצה לבצע יותר מפקודה בודדת במקרה שהביטוי מתקיים, נוכל לפתוח בלוק בעזרת סוגריים מסולסלות, ולכתוב בו מספר פקודות.
לדוגמא, נביט בתוכנית הבאה:

```
#include <stdio.h>

int main()
{
	int x;
	printf("Please enter your exam grade: ");
	scanf("%d", &x);
	if (x < 55) printf("You're a l-o-o-o-s-e-r :-)\n");
	if (55 <= x && x < 85) printf("Not bad!\n");
	if (x >= 85 && x <= 100)
	{
		printf("Excellent!!!\n");
		printf("Your grade is %d.\n", x);
	}
	return 0;
}
```

בתוכנית שלוש פקודות if. בפקודה השלישית אנו מעוניינים לבצע שתי פקודות במקרה שהתנאי של ה-if מתקיים. לפיכך, מיד אחרי ה-if, מבלי שאנו שמים נקודה פסיק בסוף שורת ה-if, אנו פותחים בלוק בעזרת סוגריים מסולסלות, ובתוכו כותבים את הפקודות שאנו רוצים שיתבצעו.

ישנן מספר טעויות נפוצות הקשורות לפקודה if. כדי להימנע מהטעויות רצוי להכיר אותן, ולכן נציג כעת את הטעויות הנפוצות ביותר.

דוגמא ראשונה:

```
#include <stdio.h>

int main()
{
      int x, y;
      scanf("%d%d", &x, &y);
      if (x = y) printf("The 2 numbers are equal\n");
      return 0;
}
```

הבעיה בקטע קוד זה: בתנאי ה-if השתמשנו באופרטור ההשמה =, ולא באופרטור ההשוואה ==. אי לכך, בכל מקרה שבו y שונה מ-0, יודפס על המסך ששני המספרים זהים, ללא קשר האם הם זהים או לא. בנוסף לכך, ערכו של x יידרס ויוחלף בערכו של y.

דוגמא שנייה:

```
#include <stdio.h>

int main()
{
      int x, y;
      scanf("%d%d", &x, &y);
      if (x == y)
            printf("The 2 numbers are equal.\n");
            printf("x equals to y.\n");
      return 0;
}
```

כוונת כותב הקטע הייתה שבמידה וערכו של x שווה לערכו של y, יודפסו שתי שורות שיציינו עובדה זו. אולם, זה לא מה שקורה בפועל. מכיוון שלא תחמנו את הקטע בסוגריים מסולסלות, תודפס בכל מקרה על המסך המחרוזת "x equals to y.".

ניתן לרשום את קטע קוד זה בצורה הבאה, שמבהירה את הבעיה:

```
#include <stdio.h>

int main()
{
      int x, y;
      scanf("%d%d", &x, &y);
      if (x == y) printf("The 2 numbers are equal.\n");
      printf("x equals to y.\n");
      return 0;
}
```

קטע קוד זה זהה לקודם, אולם כעת אנו יכולים לראות שרק ה-printf הראשון שייך לפקודת ה-if.

פתרון לבעיה זו הינו חסימת הקטע בסוגריים מסולסלות, בצורה הבאה:

```
#include <stdio.h>

int main()
{
      int x, y;
      scanf("%d%d", &x, &y);
      if (x == y)
      {
            printf("The 2 numbers are equal.\n");
            printf("x equals to y.\n");
      }
      return 0;
}
```

דוגמא שלישית:

```
#include <stdio.h>

int main()
{
      int x, y;
      scanf("%d%d", &x, &y);
      if (x == y);
      {
            printf("The 2 numbers are equal.\n");
            printf("x equals to y.\n");
      }
      return 0;
}
```

במקרה זה שמנו ; לאחר פקודת ה-if. שתי פקודות ה-printf יתבצעו בכל מקרה, ללא קשר להתקיימות הביטוי שבתוך ה-if. למעשה מה שאמרנו על ידי הנקודה פסיק, זה: "אם x שווה ל-y, אל תבצע כלום." אחרי זה אנו פותחים בכל מקרה בלוק, מדפיסים שתי שורות, וסוגרים את הבלוק.

לפקודת if מבנה כללי יותר מזה שהצגנו. המבנה הכללי של הפקודה הוא כלהלן:

```
if (expression)
{
      ...
}
[else
{
      ...
}]
```

המילה השמורה else מאפשרת לנו לקבוע פעולה שתתבצע במידה וביטוי ה-if איננו נכון.
הסוגריים המלבניות באות לציין כי ה-else הוא חלק אופציונלי של הפקודה, ואיננו חייב להופיע בכל פקודת if.
דוגמא לשימוש במבנה הכללי של if:

```
if (i > 10)
{
      printf("i is bigger than 10\n");
}
else
{
      printf("i is smaller than 10\n");
}
```

אופרטור ההתניה :?

אופרטור זה הוא למעשה דרך מקוצרת ליצירת הסתעפות. בכל מקום בו אנו משתמשים באופרטור זה, ניתן להחליפו במשפט if מקביל. עם זאת, ישנם מקרים בהם נוח להשתמש באופרטור זה במקום ב-if.
תחביר השימוש באופרטור:

```
(expression) ? expression1 : expression2;
```

משמעות האופרטור: הביטוי expression ייבדק. במידה והוא אמת, יבוצע expression1. במידה והביטוי הוא שקר, יבוצע expression2.

דוגמא:
x, y, z הם משתנים שהוגדרו קודם לכן בתוכנית.

```
z = x > y ? x : y;
```

z מקבל את הערך של הגדול מבין x ו-y.

משפט switch

כאשר אנו רוצים ליצור התניה, והביטוי שאנו רוצים לבדוק הינו מטיפוס מספר שלם או תו, נעדיף לעיתים להשתמש במילה השמורה switch. switch מספקת פונקציונליות דומה לזו שמספקת הפקודה if, אולם לעיתים קוד שנכתב בעזרת switch קריא יותר.
המבנה של משפט switch:

```
switch (expression)
{
      case VALUE1:
            ...
            break;
      case VALUE2:
            ...
            break;
      ...
      case VALUEn:
            ...
            break;

      default:
            break;
}
```

expression הוא ביטוי שערכו חייב להיות int או char בלבד.
VALUE1, VALUE2, ..., VALUEn הם ערכים שהביטוי expression יכול להחזיר.
ביצוע התוכנית קופץ אל הקטע המייצג את ערכו של הביטוי. default הוא קטע המתבצע במידה וערך ה-expression איננו אף אחד מבין VALUE1, VALUE2, ..., VALUEn.
לאחר שקפצנו אל קטע לביצוע, הפקודה break מציינת את סיומו.

דוגמא:

```
#include <stdio.h>

int main()
{
	int i;
	printf("Please enter number between 1 to 3: ");
	scanf("%d", &i);

	switch(i)
	{
		case 1:
			printf("You entered 1.\n");
			break;
		case 2:

		case 3:
			printf("You entered 2 or 3.\n");
			break;
		default:
			printf("ERROR: Illegal input.\n");
	}

	return 0;
}
```

כאשר נריץ את התוכנית, היא תחכה לקלט מהמשתמש.
במידה והמשתמש יקליד את המספר 1, יודפס "You entered 1". במידה והמשתמש יקליד את המספרים 2 או 3, יודפס "You entered 2 or 3". אם המשתמש יקליד מספר אחר תודפס הודעת שגיאה.
נשים לב שאחרי "case 2:" לא השתמשנו במילה השמורה break, ולכן הביצוע במקרה שהמשתמש מקיש 2, ממשיך אל "case 3:", ונעצר רק כאשר מגיעים אל הפקודה break הנמצאת שם.

תרגילים

תרגיל 1

כתוב תוכנית שתקבל מהמשתמש שלושה מספרים, ותציג את הגדול מביניהם על המסך.

תרגיל 2

כתוב תוכנית המקבלת מספר מהמשתמש ומחזירה את הערך המוחלט שלו.

פתרונות

פתרון 1

מטרת תרגיל זה היא לתרגל שימוש בפקודה if.

```
#include <stdio.h>

int main()
{
	int x, y, z;
	int max_num;
	printf("Please enter 3 numbers: ");
	scanf("%d%d%d", &x, &y, &z);
	if (x >= y && x >= z) max_num = x;
	else if (y >= x && y >= z) max_num = y;
	else if (z >= x && z >= y) max_num = z;
	printf("The biggest number is %d\n", max_num);
	return 0;
}
```

פתרון 2

ניתן לפתור גם תרגיל זה בעזרת תנאי if. אנו נציג פתרון המשתמש באופרטור : ?.

```
#include <stdio.h>

int main()
{
	int num;
	printf("Please enter a number: ");
	scanf("%d", &num);
	num = (num > 0) ? num : -num;
	printf("abs(num) = %d\n", num);
	return 0;
}
```

פונקציות מתמטיות

במקרים רבים על תוכניות לבצע חישובים מתמטיים - פעולות פשוטות כגון חיבור או חיסור מספרים, וכן לעיתים אף פעולות מתוחכמות יותר, כגון הוצאת שורש או חישוב לוגריתם.
כפי שכבר ראינו פעולות החשבון הבסיסיות הינם חלק אינטגרלי משפת C. אנו יכולים להשתמש באופרטורים של השפה כדי לבצע פעולות חיבור, כפל, חילוק, חיסור ועוד.
שפת C תומכת גם בפעולות מתמטיות מורכבות יותר, והיא מכילה ספריית פונקציות המיועדת למטרה זו.
על מנת להשתמש בפונקציות המתמטיות שבשפת C, עלינו להוסיף ספרייה זו בתחילת התוכנית, על ידי ההצהרה:

```
#include <math.h>
```

ספריה זו מכילה פונקציות מתמטיות רבות.
הטבלה הבאה מכילה סיכום של הפונקציות המרכזיות, הפעולה שהן מבצעות והטיפוס המוחזר:

פונקציה ב-C	פעולה מתמטית	טיפוס מוחזר
sqrt(x)	$\sqrt{x}$	double
pow(x, y)	x^y	double
exp(x)	e^x	double
log(x)	$\ln x$	double
log10(x)	$\log x$	double
sin(x)	$\sin x$	double
cos(x)	$\cos x$	double
tan(x)	$\tan x$	double
abs(x)	$\lvert X \rvert$	int
fabs(x)	$\lvert X \rvert$	float

דוגמא:

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

int main()
{
	int i;
	printf("Please enter a number: ");
	scanf("%d", &i);
	printf("Absolute value of %d is %d\n", i, abs(i));
	return 0;
}
```

שלבים להתרת בעיה

כאשר אנו כותבים תוכניות אנו כותבים אותן כדי שיפתרו בעיה כלשהי, ינתחו נתונים ויגיבו בהתאם לתוצאה. אחד הכלים החשובים ביותר שעלינו לרכוש הוא היכולת להגדיר את הבעיה ולאפיין אותה בצורה כזו כך שנוכל לגרום למחשב לפתור אותה.
שלבי העבודה שלנו יהיו אלו:

1. הגדרת הבעייה - מהו הקלט, מה האילוצים, מה צריך להיות פלט התוכנית.
2. פיתוח אלגוריתם (שיטה) לפתרון הבעייה - האלגוריתם הנכתב צריך להיות בהתאם לרמה של מי שאמור להבין את האלגוריתם, בהתאם לשפת התכנות, ובהתאם לאדם עבורו מיועדת התוכנה.
3. קידוד - הפיכת האלגוריתם לקוד בשפת התכנות.

דוגמא 1

כתוב תוכנית הפותרת משוואות ריבועיות מהצורה $ax^2+bx+c=0$.
יש להתחשב בכל המקרים האפשריים (שני פתרונות, פתרון יחיד, פתרון קומפלקסי וכו').

הגדרת הבעיה:

קלט: שלושה מספרים שלמים - a, b, c.
פלט: פתרון המשוואה, או הודעה מיוחדת במקרה שקיים פתרון יחיד/אין פתרון.
סיום הפעולה: כאשר אנו מסיימים לפתור את המשוואה.

פיתוח האלגוריתם:

ניזכר ראשית במתמטיקה הקשורה לבעיה:
למשוואה זו אין פתרון אם $a,b=0$ וגם $c\neq 0$ ויש לה אינסוף פתרונות אם $a=b=c=0$.
פתרון משוואה ריבועית ניתן על ידי הנוסחה: $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$.
הביטוי b^2-4ac יסומן על ידי האות היוונית Δ (דלתא).
במידה ו-$\Delta>0$ קיימים שני פתרונות למשוואה. במידה ו-$\Delta=0$ קיים למשוואה פתרון יחיד.
במידה ו-$\Delta<0$ למשוואה פתרונות מרוכבים.

משתנים:
מספרים שלמים:
a, b, c - מקדמי המשוואה אותה אנו פותרים.
delta - משתנה שישמור את תוצאת חישוב ה-Δ.

האלגוריתם:
1. קלוט שלושה מספרים: a, b, c.
2. אם $a,b=0$ וגם $c\neq 0$ הדפס "אין פתרון" וסיים.
3. אם $a=b=c=0$ הדפס "אינסוף פתרונות" וסיים.
4. אם $a=0$ פתור את המשוואה כמשוואה ממעלה ראשונה וסיים.
5. חשב את Δ.
6. אם $\Delta>0$ חשב את שני הפתרונות של המשוואה, הצג אותם וסיים.
7. אם $\Delta=0$ חשב את הפתרון היחיד, הצג אותו וסיים.
8. אם $\Delta<0$ חשב את שני הפתרונות המרוכבים של המשוואה, הצג אותם וסיים.

שלב הקידוד:

תוכנית זו ארוכה יותר מהתוכניות שהודגמו עד כה, ולכן נפתח אותה בשני שלבים -
שלבים
המומלצים בכל פעם שכותבים תוכנית גדולה:
בשלב הראשון נכין את המבנה הכללי של התוכנית, ונמלא אותו בהערות, שאומרות מה אנחנו מתכננים לעשות בכל שלב, על מנת ליצור תוכנית עבודה מאורגנת, ולדאוג שנזכור לטפל בכל המצבים. בשלב השני נכתוב את הקוד בין ההערות.

שלב 1:

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

int main()
{
      int a, b, c;
      int delta;
      printf("Please enter a, b, c: ");
      scanf("%d%d%d", &a, &b, &c);

      /* Check if there are solutions */

      /* Infinite number of solutions */

      /* One-degree equation? */

      /* Calculate delta */

      /* If delta is positive - there are 2 solutions */

      /* If delta is zero - there is one solution */

      /* If delta is negative - the solutions belongs to
         complex domain */

      return 0;
}
```

כתבנו מעט מאוד קוד, אולם כתבנו קוד המציין מה השלבים שנצטרך לבצע כדי לפתור את הבעיה.
כעת נקודד את התוכנית.
מומלץ לקורא לעצור כעת ולנסות לפתור את הבעיה בעצמו, ורק לאחר מכן לעיין בפתרון.

שלב 2:

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

int main()
{
	int a, b, c;
	int delta;
	double real, img;
	printf("Please enter a, b, c: ");
	scanf("%d%d%d", &a, &b, &c);

	/* Check if there are solutions */
	if (a == 0 && b == 0 && c != 0)
	{
		printf("No solution.\n");
		return 0;
	}

	/* Infinite number of solutions */
	else if (a == 0 && b == 0 && c == 0)
	{
		printf("Infinite number of solutions.\n");
		return 0;
	}

	/* One-degree equation? */
	else if (a == 0)
	{
		printf("X = %.2lf\n", (double)-c / b);
		return 0;
	}

	/* Calculate delta */
	delta = b*b - 4*a*c;

	/* If delta is positive - there are 2 solutions */
	if (delta > 0)
	{
		printf("X1 = %.2lf\n", (-b + sqrt(delta))/(2*a));
		printf("X2 = %.2lf\n", (-b - sqrt(delta))/(2*a));
		return 0;
	}

	/* If delta is zero - there is one solution */
	else if (delta == 0)
	{
		printf("X = %.2lf\n", (double)-b/(2*a));
		return 0;
	}


	/* If delta is negative - the solutions belongs to
```

```
         complex domain */
      else
      {
            real = (double)(-b)/(2*a);
            img = (double)sqrt(-delta)/(2*a);

            /* x1 */
            printf("X1 = ");
            if (real != 0)
            {
                  printf("%.2lf", real);
                  if (img > 0) printf(" + ");
                  else if (img < 0) printf(" - ");
                  if (img == 1 || img == -1) printf("i\n");
                  else printf("%.2lfi\n", img);
                  return 0;
            }
            else
            {
                  if (img == 1) printf("i\n");
                  else if (img == -1) printf("-i\n");
                  else printf("%.2lfi\n", img);
            }

            /* x2 */
            img = -img;
            printf("X2 = ");
            if (real != 0)
            {
                  printf("%.2lf", real);
                  if (img > 0) printf(" + ");
                  else if (img < 0) printf(" - ");
                  if (img == 1 || img == -1) printf("i\n");
                  else printf("%.2lfi\n", img);
                  return 0;
            }
            else
            {
                  if (img == 1) printf("i\n");
                  else if (img == -1) printf("-i\n");
                  else printf("%.2lfi\n", img);
                  return 0;
            }
      }
      return 0;
}
```

כאשר באנו לממש את האלגוריתם, אנו עוברים שורה שורה על האלגוריתם, ממשים אותה בקוד, ואז עוברים אל השורה הבאה.
ניתן לראות כי כל הפעולות פשוטות יחסית, מלבד הטיפול במספרים מרוכבים. הסיבוך בהצגת המספרים המרוכבים נובע בעיקר משיקולי אסטטיות - למשל, אנו רוצים להדפיס i, ולא 1.00i במקרה שהחלק המדומה של המספר שווה ל-1, וכו'.

המשתנים img, real לא הופיעו באלגוריתם. משתנים אלו הוספו לתוכנית כדי להפוך אותה לקריאה יותר, ולמנוע מעט שכפול קוד. נוכחותם איננה חיונית - ניתן היה לכתוב את התוכנית גם בלעדיהם.
כאשר אנו באים לתכנן את האלגוריתם איננו מפרטים עד לשלב האחרון כיצד התוכנית תמומש. לשם כך קיים שלב הקידוד. המשתנים והפעולות שאנו מגדירים באלגוריתם צריכים להיות כלליים לפתרון השאלה, ואינם תלויים במימוש זה או אחר.

דוגמא 2

בדוגמא זו נשתמש בכלי שעדיין לא נלמד - לולאות. עם זאת, הדגש בדוגמא זו איננו על השימוש בלולאות, אלא על פיתוח האלגוריתם. הסבר מפורט על לולאות יופיע בהמשך המסמך.

הבעיה:
מוכר בחנות עומד ליד הקופה. כאשר הוא מרוויח 1000 ש"ח הוא מסיים את יום העבודה והולך הביתה.
אנו צריכים לכתוב תוכנית שתגיד לו מתי הוא יכול ללכת.

הגדרת הבעיה:

קלט: סדרת מספרים שלמים.
פלט: הסכומים המצטברים.
סיום: כאשר הסכום גדול מ-1000.

פיתוח האלגוריתם:

משתנים :
מס' שלמים:
X - המספר הבא
Sum - הסכום המצטבר

האלגוריתם:
1. אפס את Sum.
2. עשה:
3. קרא את המספר הבא לתוך X.
4. הוסף את X ל-Sum .
5. הדפס את Sum
6. כל עוד $Sum \leq 1000$

השורות 3-5 מתבצעות כל עוד התנאי שבשורה 6 אינו מתקיים.

קיימת תקלה באלגוריתם התוכנית כפי שהוצג.
התקלה - התוכנית לא מוציאה שום פלט. המוכר המסכן לא הגיע הביתה עד עצם היום הזה.
נהוג לכנות תקלות מסוג זה בשם באגים - bugs.
האלגוריתם שכתבנו עובד, אולם הוא לא עושה את מה שהתכוונו שיעשה.

נציג אלגוריתם משופר שיפתור את הבעיה:

1. אפס את Sum.
2. עשה:
3. קרא את המספר הבא לתוך X.
4. הוסף את X ל-Sum .
5. הדפס את Sum
6. כל עוד $Sum \leq 1000$
7. הדפס "לך הביתה".

שלב הקידוד:

קידוד תוכנית זו פשוט יחסית. אנו משתמשים בקידוד בצמד מילים חדש - do... while.
המחשב מבצע את הבלוק שבין הסוגריים המסולסלות, כל עוד הביטוי שבתוך ה-while מתקיים.

```
#include <stdio.h>
int main()
{
	int x, sum; /* x - input var., sum - total numbers */
	sum = 0;
	do
	{
		scanf("%d", &x);
		sum += x;
		printf("%d\n", sum);
	}
	while (sum <= 1000);

	printf("Go Home!!!\n");
	return 0;
}
```

קבועים ושמות נרדפים

קבועים

לעיתים קרובות נרצה להגדיר ערך קבוע - להעניק למספר כלשהו שם מיוחד.
למשל, ייתכן שנרצה לשייך למספר 3.142 את השם PI.

נרצה להגדיר ולהשתמש בקבועים עקב שתי סיבות עקריות:
1. קריאות התוכנית - שימוש במילים המציינות ערך מספרי הרבה יותר קריא משימוש במספרים באותם מקומות. למשל - נניח שאנחנו כותבים תוכנית לניהול מספר הנוסעים בכל מעלית, שימוש בשם קבוע כגון MAX_USERS_ALLOWED הוא הרבה יותר ברור משימוש, למשל, במספר 7 בכל אותם מקומות.
2. תחזוק מהיר - אם ערך מספרי מופיע בתוכנית שלנו פעמים רבות, וישנה אפשרות שהוא ישתנה ביום מן הימים, נעדיף לשמור אותו כקבוע.
נניח, לדוגמא, כי הערך 1000 מציין את המשקל אותו המעלית מסוגלת לסחוב. נניח כי ערך זה מופיע 20 פעמים בתוכנית שלנו. במקרה והמשקל ייגדל ל-1200 בדור החדש של המעליות, נאלץ לשנות את הקוד שלנו ב-20 מקומות. אם לעומת זאת, נשתמש בקבוע אחד, ובכל 20 המקומות נרשום את שם הקבוע, כל שיהיה עלינו לעשות הוא לשנות את ערכו של הקבוע, והתוכנית תעבוד כשורה.

שפת C מאפשרת לנו להגדיר קבועים בשני דרכים: בעזרת ההוראה #define ובעזרת המילה השמורה const.

המבנה התחבירי של ההוראה #define:

```
#define CONST_NAME Expression
```

כאשר CONST_NAME הוא שם הקבוע אותו אנו רוצים להגדיר, ו-Expression הוא הביטוי המתאים לקבוע זה.
למשל:

```
#define PI 3.142
```

הוראת #define איננה מסתיימת בנקודה פסיק. כמו כן לא השתמשנו באופרטור = כדי לקבוע את ערכו של PI, אלא כתבנו מייד לאחריו את הביטוי שאנו רוצים ש-PI ייצג.

השימוש בפקודה #define אינו יוצר משתנה בשם PI. תתבצע פעולת החלפה פשוטה - בכל מקום בתוכנית בו תופיעה המילה PI, היא תוחלף על ידי המספר 3.142.

נביט בדוגמא. קטע הקוד הבא מאפשר למשתמש להמשיך רק אם הוא בן +18:

```
#include <stdio.h>

#define MIN_AGE 18

int main()
{
	int iAge;
	printf("Please enter your age: ");
	scanf("%d", &iAge);
	if (iAge < MIN_AGE)
	{
		printf("You are too young. Sorry...\n");
		return 1;
	}

	printf("Welcome, you can use the program.\n");
	return 0;
}
```

כפי שניתן לראות, הגדרנו עד כה את כל הקבועים בעזרת שמות המורכבים מאותיות גדולות - Uppercase. למרות שניתן לתת לקבועים שמות המורכבים מאותיות קטנות, מקובל כשמתכנתים ב-C להשתמש רק באותיות גדולות, וזאת על מנת שיהיה קל להבחין בין קבועים לבין משתנים.

דרך שניה להגדיר קבועים בשפת C היא בעזרת המילה השמורה const.
המבנה התחבירי של הפקודה:

```
const <type> <name> = <value>;
```

כאשר אנו משתמשים בפקודה זו, אנו יוצרים משתנה, אולם אומרים ל-C כי ערכו קבוע ואינו ניתן לשינוי. נסיון לשנות את ערכו יגרור שגיאת קומפילציה.

דוגמא לשימוש:

```
const int PI = 3.142;
```

לולאות

לולאות הן אחד הכלים החשובים ביותר של שפת C. לולאות הן בעצם השיטה שלנו לומר למחשב לבצע את אותו קטע קוד מספר פעמים, עד שיתקיים תנאי מסוים.
נראה מיד כי השימוש בלולאות יגדיל במידה רבה את התחכום של התוכניות שנכתוב.

שפת C מכילה מספר סוגים של לולאות. מתכנתים מתחילים מתבלבלים לעיתים באיזה מסוגי הלולאות צריך להשתמש בכל מקרה. ננסה לתת קווים מנחים לבחירה בכל אחד מסוגי הלולאות, ולהראות למה הוא עדיף על הסוגים האחרים באותו מקרה.

לולאת for

התחביר של לולאת for:

```
for ([<initialize>]; [<condition>]; [<step>])
{
      /* do something */
}
```

initialize, condition ו-step הם שלושה ביטויים.
כאשר אנו משתמשים בפקודת for, ראשית מתבצע הביטוי initialize. לאחר מכן נבדק ערכו של הביטוי condition. אם ערכו של ביטוי זה הוא אמת, נבצע את הפקודות שבין הסוגריים המסולסלות.
לאחר מכן, נבצע את הביטוי step, ונחזור לבדוק שוב את condition. בצורה כזו נמשיך: ביצוע הקוד שבין הסוגריים המסולסלות, ביצוע הביטוי step ובדיקת ערכו של condition.
אם לא קיימים סוגריים מסולסלות אחרי ה-for, תבוצע הפקודה שאחרי for.

על מנת לפשט את ההסבר, נביט דוגמא:

```
#include <stdio.h>

int main()
{
      int i;
      for (i = 1; i <= 10; i++)
      {
            printf("%d\n", i);
      }

      return 0;
}
```

נביט בלולאה ה-for. כדי להבין מה היא מבצעת, נקרא אותה בצורה הבאה:
אתחל את i להיות 1. כל עוד i קטן או שווה ל-10, בצע את הקטע שבין הסוגריים המסולסלות.
כל פעם, לאחר שביצעת את הקטע שבין הסוגריים המסולסלות, קדם את i ב-1.

ניתן לראות כי הלולאה תדפיס את כל המספרים בין 1 ל-10, כל אחד בשורה נפרדת.

דוגמא נוספת: התוכנית הבאה מדפיסה את כל המספרים בין 1 ל-100, המתחלקים ב-3 או ב-7 ללא שארית.

```
#include <stdio.h>

int main()
{
	int i;
	for (i = 1; i <= 100; i++)
	{
		if (i % 3 == 0 || i % 7 == 0) printf("%d\t", i);
	}
	putchar('\n');

	return 0;
}
```

נרצה להשתמש בלולאת for כאשר אנו רוצים שקטע קוד מסוים ירוץ מספר קבוע של פעמים.
לדוגמא: בדוגמא הראשונה, קטע הקוד רץ 10 פעמים, ובדוגמא השנייה קטע הקוד רץ 100 פעמים.
מספר הפעמים לא חייב להיות ידוע כאשר אנו כותבים את התוכנית, אבל כל עוד הוא קבוע, לרוב יהיה נוח להשתמש בלולאת for.

מה עושה התוכנית הבאה?

```
#include <stdio.h>

int main()
{
	int i;
	for (i = 1; i <= 10; i++);
	{
		printf("%d\n", i);
	}

	return 0;
}
```

לכאורה היא מבצעת בדיוק את מה שביצעה הדוגמא הראשונה - מדפיסה על המסך את כל המספרים בין 1 ל-10. אולם נשים לב ל-; (נקודה פסיק) המופיעים אחרי ביטוי ה-for. בדומה למה שראינו לגבי if, זה כאילו אמרנו:

```
for (i = 1; i <= 10; i++) /* do nothing */;
```

כלומר - הלולאה אכן מתבצעת 10 פעמים, אולם היא איננה מבצעת דבר מלבד שינוי ערכו של i.

בדוגמא שלנו, לאחר מכן יודפס מספר אחד, והוא 11, ואז התוכנית תסתיים.

כל אחד מהפרמטרים של לולאת for הוא אופציונאלי.

ניתן למשל לא לכתוב תנאי אתחול (לדוגמא אם המשתנה i אותחל כבר קודם):

```
for (; i < 10; ++i)
{
     /* do something */
}
```

ניתן לא לרשום אף ביטוי בלולאת ה-for. נקבל לולאה שתרוץ לנצח:

```
for (;;)
{
     /* do something */
}
```

לולאת while

התחביר של לולאת while:

```
while (condition)
{

}
```

עם הכניסה ללולאה, נבדק הביטוי condition. אם ערכו אמת, נבצע את הקטע שבין הסוגריים המסולסלות. בסיום הקטע נבדוק שוב את התנאי condition, ואם הוא ממשיך להתקיים נחזור על התהליך - נבצע את הפקודות שבסוגריים המסולסלות ונחזור שוב לבדוק את condition.
משמעות לולאה זו היא בעצם "כל עוד condition מתקיים, בצע את הביטוי שבין הסוגריים המסולסלות".

נעדיף להשתמש בלולאה זו ולא בלולאת for, כאשר אנו רוצים שקטע קוד כלשהו יתבצע כל עוד מתקיים תנאי מסוים, ולא בהכרח מספר קבוע של פעמים.
למשל: "כל עוד קיים קלט, נתח אותו".

הדוגמא הראשונה שנביא עבור while היא הצגת המספרים בין 1 ל-10. כמו שאמרנו, פעולה כזו מתאימה יותר ללולאת for, אולם ניתן לבצע אותה גם בעזרת לולאת while. נציג דוגמא זו על מנת שיהיה ניתן לראות את ההבדלים בין מימוש משימה זו בעזרת for למימוש משימה זו בעזרת while.

```
#include <stdio.h>

int main()
{
	int i;

	i = 1;
	while (i <= 10)
	{
		printf("%d\n", i);
		i++;
	}

	return 0;
}
```

ניתן לראות שמימוש משימה זו בעזרת while מסורבל יותר ממימוש משימה זו בעזרת for.
במקום שורה אחת עבור הלולאה שהופיעה ב-for, יש לנו כאן שלוש שורות: שורה לאתחול ערכו של i, הנמצאת מחוץ ללולאה, שורת condition המכילה את התנאי להמשך הלולאה, ועוד שורה לקידום ערכו של i.

על מנת להציג דוגמאות נוספות הקשורות ל-while, נציג תכונה של הפונקציה scanf שלא הוצגה עד כה: הפונקציה scanf מחזירה ערך. הערך שהיא מחזירה מציין כמה משתנים קראה scanf בהצלחה.

נביט בקוד הבא:

```
	int x, y, i1, i2, i3;
	x = scanf("%d%d", &i1, &i2);
	y = scanf("%d", &i3);
```

אם כל המשתנים יקראו על ידי scanf בהצלחה, אזי בסיום קטע קוד זה, ערכו של x יהיה שווה 2 (הפקודה scanf קראה שני ערכים בהצלחה) ואילו ערכו של y יהיה 1 (הפקודה scanf קראה ערך אחד בהצלחה).
אם לא נקראו שדות, scanf מחזירה 0. אם הגענו אל סוף הקלט, scanf מחזירה EOF (-1).

ננצל כעת מידע זה. נרצה לקלוט קלט מספרי, ולנתח אותו, כל עוד קיימים עוד מספרים.
נעשה זאת בצורה הבאה:

```
int num;
while (scanf("%d", &num) > 0)
{
      /* do something */
}
```

בלולאה זו אנו אומרים: כל עוד יש קלט, בצע איתו פעולה כלשהי, ואז קרא את הערך הבא מהקלט.

<u>דוגמא</u>

הדוגמא הבאה תקלוט מספרים מהמשתמש. כאשר נגמרים המספרים בקלט, התוכנית תדפיס את המספר הגדול מביניהם ואת סכומם.
מציאת המספר הגדול נעשית בדרך הבאה: אנחנו שומרים משתנה max, שיכיל בכל רגע את הערך הגדול ביותר שנקלט עד כה. כאשר נקלט מספר חדש נשווה אותו ל-max. אם הוא גדול יותר, נשנה את ערכו של max לערכו של מספר זה.

```
#include <stdio.h>

int main()
{
      int max, sum, current;

      /* read first number */
      if (scanf("%d", &current) < 1)
      {
            printf("ERROR: no input\n");
            return 0;
      }

      /* initialize values */
      max = sum = current;

      /* all the rest */
      while (scanf("%d", &current) > 0)
      {
            max = (max > current) ? max : current;
            sum += current;
      }
      printf("Maximum is %d\nTotal sum is %d\n", max, sum);
      return 0;
}
```

דוגמא

נכתוב תוכנית המקבלת מספרים, המייצגים שניות. השניות מציינות את הזמן שעבר מאז הרגע הקודם בו הוכנס קלט לתוכנית. נניח כי הזמן מתחיל בשעה 12 בלילה. נרצה שכל פעם שמתקבל קלט התוכנית תדפיס מה השעה הנוכחית. התוכנית תיעצר כאשר נגיע לצהריים.
לדוגמא:

קלט	פלט
7200	2:0:0
65	2:1:5
65	2:2:10

קיימות שתי שיטות לפתרון:

1. שיטה אחת היא להוסיף את המספר לשניות, להגדיל את הדקות כל עוד מספר השניות גדול מ-60, תוך כדי הורדה של מספר השניות. לאחר מכן להגדיל את השעות כל עוד מספר הדקות גדול מ-60, ולהוריד תוך כדי כך את מספר הדקות.
2. השיטה השנייה היא שמירת מספר השניות הכולל וחישוב כל פעם מחדש של הדקות, השניות והשעות של השעה הנוכחית. זו השיטה בה נשתמש.
מספר השניות יחושב: TotalSeconds%60, מספר הדקות: (TotalSeconds/60)%60
ומספר השעות: TotalSeconds/3600.

משתנים הדרושים לפתרון הבעיה:
סכום כולל - TotalSeconds
שעה - hours
דקה - minutes
שניות - seconds

שלב הקידוד:

```
#include <stdio.h>
int main()
{
	int hours = 0, minutes, seconds;
	int TotalSeconds = 0;
	int next;

	while (hours < 12)
	{
		scanf("%d", &next);
		TotalSeconds += next;
		hours = TotalSeconds / 3600;
		minutes = (TotalSeconds / 60) % 60;
		seconds = TotalSeconds % 60;
		printf("The time is %d:%d:%d\n", hours, minutes,
			seconds);
	}
	return 0;
}
```

לולאת do...while

לולאה זו אומרת "עשה... כל עוד התנאי מתקיים".
כלומר - קודם כל גוף הלולאה מתבצע, ורק לאחרי הביצוע הראשון אנו בודקים האם תנאי הלולאה מתקיים או לא.
לולאה כזו תרוץ תמיד פעם אחת לכל הפחות.
התחביר של לולאת do...while:

```
do
{

} while (condition);
```

תנאי הלולאה נבדק בסיום כל ריצה של הלולאה.

לולאה זה פחות שימושית מבין הלולאות, מכיוון שלרוב נרצה לבדוק שתנאי כלשהו מתקיים, לפני שנכנס ללולאה. אם זאת, נשתמש לעיתים גם בלולאה זו.

נדגים שוב את הדפסת המספרים 1 עד 10 על המסך, הפעם בעזרת do... while.

```
#include <stdio.h>

int main()
{
	int i;

	i = 1;
	do
	{
		printf("%d\n", i);
		i++;
	} while (i <= 10);

	return 0;
}
```

break, continue

break, continue הן שתי מילים שמורות, המאפשרות שליטה על ביצוע הלולאה.
ראשית נציין מינוח מקובל: מעבר אחד על גוף הלולאה - מתחילת סוגריים מסולסלות עד סופן, נקרא איטרציה.

כעת: המילה השמורה continue, כאשר אנו משתמשים בה בתוך לולאה, אומרת לשפת C לסיים את האיטרציה הנוכחית של הלולאה, ולעבור אל האיטרציה הבאה.

נביט לדוגמא בקוד הבא:

```
#include <stdio.h>

int main()
{
	int i;
	for (i = 1; i <= 10; i++)
	{
		if (i == 7) continue;
		printf("%d\n", i);
	}

	return 0;
}
```

תוכנית זו תדפיס את כל המספרים בין 1 ל-10, מלבד המספר 7 עליו תדלג.

המילה השמורה break אומרת ל-C לסיים את הלולאה הנוכחית באופן מיידי, ולעבור לקוד שאחרי הלולאה. לדוגמא:

```
#include <stdio.h>

int main()
{
	int i;
	for (i = 1; i <= 10; i++)
	{
		if (i == 7) break;
		printf("%d\n", i);
	}

	return 0;
}
```

תוכנית זו תדפיס את המספרים 1 עד 6, ותעצור.

סיום

בזאת מסמך זה מגיע אל סיומו. הצגנו במסמך את הבסיס של שפת C, וחלק קטן מהכלים אותה השפה מציעה. שפת C מכילה אלמנטים רבים שלא הוצגו במסמך זה, וכן יש מקום להעמיק עוד בנושאים שהוצגו. נושאים שלא כוסו במסמך כוללים: פונקציות, מערכים ומצביעים, מחרוזות, קבצים ונושאים מתקדמים נוספים.
אני מקווה עם זאת, שהצלחתי במסמך זה לעניין אותך הקורא בשפת C, ולהציג בבהירות את הרעיונות הבסיסיים שלה, את צורת החשיבה הנדרשת כאשר באים לתכנת בשפת C.

EOF