משתנים גלובלים

- הגדרת משתנה גלובלי.
- פקודת define.

טווח הכרה של משתנים

- אנחנו יודעים שכשאנחנו מגדירים משתנה, הוא מוכר רק בתחום הבלוק בו הוא מוגדר.
- למשל, אם נגדיר משתנה בתוך פונקציה, אי אפשר יהיה לגשת אליו מפונקציות אחרות או מה-main.
- איזור הקוד בו מוכר המשתנה נקרא **טווח ההכרה של המשתנה (Scope)**.

משתנים גלובלים

- ניתן להגדיר משתנים שטווח ההכרה שלהם יהיה כל התוכנית. משתנים כאלה נקראים **משתנים גלובלים**.
- משתנה גלובלי מוגדר לפני ה-main ולכן גם ה-main וגם כל פונקציה אחרת באותה תוכנית יכולים לגשת אליו.

```
#include <stdio.h>

int x;
void f();

int main()
{
    x = 8;
    f();
    printf("%d", x);
}

void f()
{
   x = 10;
}
```

המשתנה מוגדר מחוץ ל-main ולכן הוא גלובלי – מוכר בכל התוכנית.

חוקי, המשתנה מוכר בתוך ה-main

ערכו של x הוא 10, הפונקציה שינתה את אותו ה-x שמודפס.

חוקי, המשתנה מוכר בתוך הפונקציה

```
#include <stdio.h>

int x;

int main()
{
    printf(“%d”, x);
}
```

0

שימו לב שבניגוד למשתנים מקומיים, משתנים גלובלים מאותחלים ברגע הגדרתם – המשתנים המספריים מאותחלים ל-0.

```
#include <stdio.h>

int x;

int main()
{
   int x;
   x = 9;
   printf(“%d”, x);
}
```

9

חוקי – ה-main מגדיר לעצמו משתנה מקומי בשם x, ולכן בתוך תחומי ה-main ה-x המקומי הוא המוכר, ולא הגלובלי.

משתנים גלובלים - חסרונות

- למרות הנוחות היחסית של משתנים גלובלים, כדאי להמעיט בשימוש בהם. משתנה גלובלי מקטין את העצמאות של הפונקציות והופך אותן תלויות בגורם חיצוני.
- בנוסף, קשה יותר לדבג ולשלוט בתוצאות של תוכנית שמכילה משתנים גלובלים, כיוון שכל חלק בקוד יכול לשנות אותם. כאשר הקוד ארוך ומכיל הרבה שורות, זה מצב קטסטרופלי.

משתנים גלובלים - יתרונות

- מצד שני, שימוש נכון במשתנים גלובלים יכול להיות נוח – משתנים גלובלים יהיו מאוד שימושיים אם הם יוגדרו כקבועים. כך נמנע את בעיית השליטה בשינוי המשתנה (אי אפשר לשנות אותו).
- בדר"כ נוהגים להגדיר את גדלי המערכים כמשתנים גלובלים קבועים, וכך לא צריך להעביר אותם לפונקציות.

```
#include <stdio.h>

const int N = 5;
void printArray(int[]);

int main()
{
   int a[N] = {1, 2, 3, 4, 5};
   printArray(a);
}

void printArray(int a[])
{
   int i;
   for(i = 0; i < N; i++)
         printf("%d\t", a[i]);
}
```

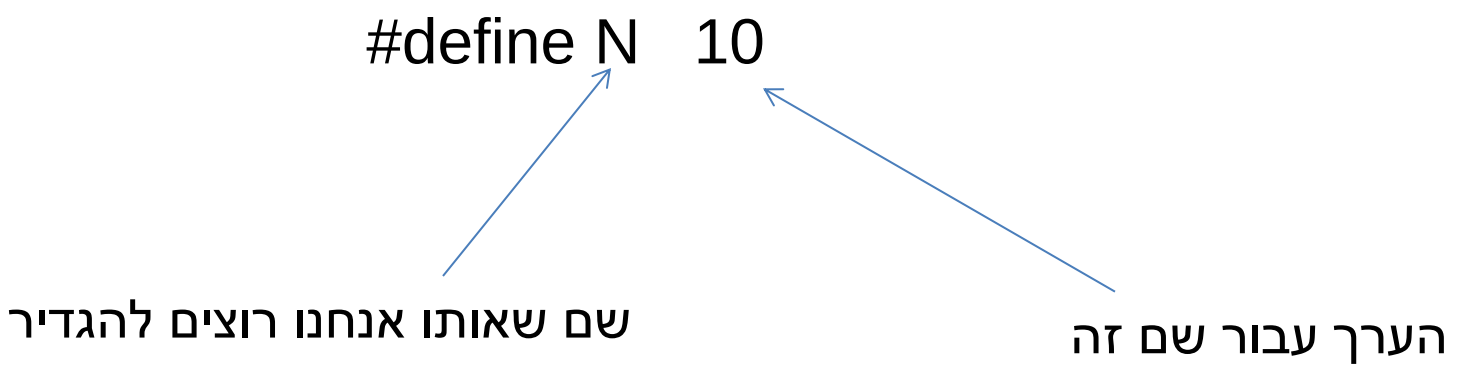
```
#include <stdio.h>

const int N = 5;
void printArray(int[]);

int main()
{
   int a[N] = {1, 2, 3, 4, 5};
   printArray(a);
}

void printArray(int a[])
{
   int i;
   for(i = 0; i < N; i++)
         printf("%d\t", a[i]);
}
```

שימו לב לשימוש שנעשה בקבוע הגלובלי, כדי ליצור קוד כללי יותר ונוח יותר להבנה.

define

- שפת C תומכת בפקודה נוספת להגדרת שמות גלובלים.
- פקודה זו נקראת #define והיא מוגדרת כפקודת קדם-קומפיילר (pre compilation), כלומר, היא מתבצעת לפני שהקומפיילר מתחיל לעבור על הקוד.
- פקודת ה-define נקראת גם מאקרו (macro)

מבנה הפקודה – הפקודה נכתבת בראש התוכנית, אחרי פקודות ה-include והיא בנויה בצורה הבאה:



למעשה, אנחנו כאילו אומרים לקומפיילר: "בכל מקום בו אתה רואה את הסימן N החלף אותו ב-10"

```
#include <stdio.h>
#define N   5

void printArray(int[]);

int main()
{
   int a[N] = {1, 2, 3, 4, 5};
   printArray(a);
}

void printArray(int a[])
{
   int i;
   for(i = 0; i < N; i++)
         printf("%d\t", a[i]);
}
```

שימו לב שה-N מתפקד פה כקבוע גלובלי.

define - יתרונות

- ההבדל העקרוני בין פקודת המאקרו למשתנה גלובלי, הוא שמאקרו הוא לא משתנה, כלומר, לא תופס מקום בזיכרון. אולם כיום זה יתרון פחות רלבנטי.
- לפעמים נוח יותר להשתמש במאקרו כיוון שהגדרתו ברורה לעין ומסודרת יותר.

תרגיל

- כתבו פונקציה שחתימתה –

int isTwoDigits(int)

- הפונקציה תקבל כפרמטר מספר שלם, ותחזיר TRUE אם המספר בעל שתי ספרות ו-FALSE אם לא.
- השתמשו במאקרו כדי להגדיר את המילים TRUE ו-FALSE.
- כתבו גרסה נוספת של התוכנית תוך שימוש במשתנה גלובלי להגדרת המילים.