מבוא למדעי המחשב

תירגול 4: משתנים בוליאניים ופונקציות

© TemplatesWise.com

משתנים בוליאניים

© TemplatesWise.com

ערכי אמת

- מבחינים בין שני ערכי אמת: true ו-false
- לכל מספר שלם ניתן להתאים ערך אמת:
  - הערך 0 הוא בעל ערך אמת false
  - כל ערך אחר הוא בעל ערך אמת true

0	false
1, 5, -200, 'a'	true

ערכי אמת - דוגמה

```
if (0) {
  printf("this will never be printed");
}
if (-30) {
  printf("this will always be printed");
}
```

```
int n;
scanf("%d", &n);
if (n) {
   printf("n is non-zero!");
}
```

הטיפוס bool

- ב-C קיים טיפוס ייעודי לייצוג ערכי אמת
- כדי להשתמש בו יש לכלול את הספריה `stdbool.h`:
- מוגדר בה הטיפוס `bool`
- הקבוע `false` שמוגדר כ-0
- הקבוע `true` שמוגדר כ-1
- ההמרה מ-`int` ל-`bool` ולהיפך מתבצעת לפי ערכי האמת

הטיפוס bool - דוגמה

```
#include <stdbool.h>
...
int n;
scanf("%d", &n);
bool notZero = n;
if (notZero) {
   printf("n is non-zero!");
}
```

- לא מומלץ להשוות בין משתני int לערכי bool.

מה היה קורה פה?

```
if (n != true) {
   printf("n is zero");
}
```

אופרטורי השוואה

- אופרטורים בינאריים המחזירים 0 או 1 בהתאם לנכונות ההשוואה:

A שווה ל-B	`A == B`
A שונה מ-B	`A != B`
A גדול מ-B	`A > B`
A קטן מ-B	`A < B`
A גדול או שווה ל-B	`A >= B`
A קטן או שווה ל-B	`A <= B`

אופרטורים לוגיים

- אופרטורים המתייחסים לפרמטרים שלהם כערכי אמת ומחזירים 0 או 1 בהתאם לפירוט בטבלה:

מה מחזיר ?	שם הפעולה	האופרטור
היפוך ערך האמת של A (מחזיר 0 אם A מתקיים, ו-1 אם אינו מתקיים).	not A	`!A`
מחזיר 1 אם גם A וגם B מתקיימים.	A and B	`A && B`
מחזיר 1 אם **לפחות** אחד מבין A, B מתקיים.	A or B	`A || B`

דוגמאות

הביטוי	תוצאה
`!5`	0
`!!5`	1
`5 && 0`	0
`5 && 'A'`	1
`5 || (-5)`	1
`5 || 0`	1

דוגמאות בקוד

```
if (0 <= n && n <= 10) {
   printf("n is between 0 and 10");
}
```

```
if (n < 0 || n > 10) {
   printf("n smaller than 0 or larger than 10");
}
```

```
if (n % 7 == 0) {
   printf("n is divisible by 7");
}
```

מנגנון short-circuit evaluation

- החישוב של ביטוי לוגי נעצר ברגע שיודעים מה יהיה ערכו.
- מנגנון זה מתייחס לאופרטורים || ו-&&

A && B

אם **A** לא מתקיים - ברור שהתוצאה היא 0 ולכן החישוב יעצור.

A || B

אם **A** מתקיים - ברור שהתוצאה היא 1 ולכן החישוב יעצור.

דוגמאות

```
int speed = 100;
(speed > 90) || (speed < 55);
```

- אם speed>90, מוחזר 1 בלי לבדוק את התנאי השני
- תכונת הלוגיקה המקוצרת מאפשרת לשלב כמה בדיקות בו זמנית, גם כאשר בדיקה אחת תלויה בתוצאה של בדיקה קודמת:

```
if ((x != 0) && (1/x > 12))
```

בדוגמה הזו, אם x=0 לא נבצע את החלוקה בצד ימין

תרגיל 1

- מה תדפיס התוכנית הבאה?

```
#include <stdio.h>
#include <stdbool.h>
int main()
{
   int i=10;
   bool stop=false;
   while(!stop && (i-- >= 0)){
        stop=!(i%5);
        printf("%d\n",i);
   }
   printf("%d\n",i);
   return 0;
}
```

תרגיל 2 (מתוך ש.ב)

מספר הוא אי זוגי בריבוע אם הוא מכיל מספר אי זוגי של ספרות אי זוגיות.

לדוגמה: המספר 12345 הוא אי זוגי בריבוע, כיוון שהוא מכיל 3 ספרות אי-זוגיות (1,3,5)

לעומתו המספר 1234 הוא לא אי זוגי בריבוע, כיוון שהוא מכיל 2 ספרות אי זוגיות (1,3).

א. יש לכתוב תוכנית הקולטת מהמשתמש מספר חיובי הקטן (ממש) ממיליון. אם המספר הוא אי זוגי בריבוע -

- יש להדפיס:

The number is double odd: v

- אחרת, יש להדפיס:

The number is double odd: x

תרגיל 2- פתרון

```
int val, oddCounter = 0;
scanf("%d", &val);
while (val > 0){
    int digit = val % 10;
    if (digit % 2) {
        oddCounter++;
    }
    val /= 10;
}
printf("The number is double odd: %c",
            (oddCounter % 2) ? 'v' : 'x');
```

תרגיל 2 (המשך)

ב. יש לכתוב שוב את אותה התכנית, רק הפעם ללא שימוש במשפטי תנאי (if, else, switch, ?:)

רמז: השתמשו במשתנים בוליאניים

תרגיל 2- פתרון

```
int val, oddCounter = 0;
scanf("%d", &val);
while (val > 0){
    int digit = val % 10;
    if (digit % 2) {
        oddCounter++;
    }
    val /= 10;
}
printf("The number is double odd: %c",
            (oddCounter % 2) ? 'v' : 'x');
```

תרגיל 2- פתרון

```
int val, oddCounter = 0;
scanf("%d", &val);
while (val > 0){
    int digit = val % 10;
    bool isDigitOdd = digit % 2;
    oddCounter += isDigitOdd;
    val /= 10;
}
printf("The number is double odd: %c",
            (oddCounter % 2) ? 'v' : 'x');
```

תרגיל 2- פתרון

```
int val, oddCounter = 0;
scanf("%d", &val);
while (val > 0){
    int digit = val % 10;
    bool isDigitOdd = digit % 2;
    oddCounter += isDigitOdd;
    val /= 10;
}
bool isCounterOdd = oddCounter % 2;
printf("The number is double odd: %c",
            'x'+isCounterOdd*('v'-'x'));
```

מבוא לפונקציות

© TemplatesWise.com

פונקציות ב-C

- ניתן להתייחס לפונקציה כמו קופסה שחורה שמקבלת ערכים (**פרמטרים**) ומחזירה תוצאת חישוב עליהם (**ערך החזרה**).
- ראינו בעבר דוגמאות לפונקציות: printf, scanf...
- אנחנו יכולים ליצור פונקציות משל עצמנו לפי הצורך.
- יש להבדיל בין קריאה לפונקציה, שעושה שימוש בפונקציה כקופסא שחורה (כפי שעשינו עם scanf, printf), לבין הגדרת הפונקציה והמימוש שלה, שם מתבצע החישוב בפועל.

דוגמה – מימוש פונקציה

- ננתח את הפונקציה הבאה, שמקבלת שני מספרים ומחזירה את ההפרש ביניהם בערך מוחלט.
- לכל פונקציה יש חתימה:

הטיפוס המוחזר ע"י הפונקציה מופיע לפני שמה.

שם הפונקציה.

בסוגריים: רשימת הפרמטרים (מופרדים בפסיקים ביניהם).

```
float delta(float a, float b) {
  float result = a - b;
  if (result < 0)
    result = -result;
  return result;
}
```

דוגמה - המשך

- רכיבים נוספים של הפונקציה:

משתנה מקומי: קיים רק בתוך הפונקציה, בזמן שהפונקציה פועלת. "נשכח" לאחר סיום פעולת הפונקציה.

```
float delta(float a, float b) {
  float result = a - b;
  if (result < 0)
    result = -result;
  return result;
}
```

גוף הפונקציה

ההוראה `return` משמשת לקביעת **הערך המוחזר** על-ידי הפונקציה.
זה בעצם הערך שהפונקציה מחשבת.
בדוגמה אנו יכולים להחזיר את result כי ערך החזרה של delta מהטיפוס המתאים

פקודת return

- פעולת `return` מסיימת את הפונקציה הנוכחית, וההרצה נמשכת **מהמקום בו קראנו לפונקציה**.
- הערך שהועבר ל-`return` הוא הערך "שהפונקציה מחזירה", כלומר התוצאה.
- כאשר הפונקציה לא מחזירה ערך ("טיפוס" מוחזר `void`), `return` לא מקבלת ערך (וגם לא הכרחית לכתיבה).
- כאשר הפונקציה מחזירה ערך, הפקודה return עם טיפוס מתאים הכרחית בכל מסלולי החישוב.

main כפונקציה

- פונקציית ה- main() היא נקודת ההתחלה של כל תוכנית, והיא נקראת ע"י מע' ההפעלה.
- חתימת הפונקציה:

```
int main();
```

- ביצוע return עם ערך כלשהו מתוך main() מסיים את ריצת התוכנית ומחזיר את אותו ערך למע' ההפעלה.
- בד"כ נהוג:
  - החזרת 0 בסיום תקין
  - כל ערך אחר בשגיאה

המשך דוגמה – קריאה לפונקציה

```
float delta(float a, float b);
```

```
int main(){
  float x = 3.4, y = 5.7;
  float d = delta(x,y);
  printf("delta of x and y is: %f\n", d);
  d = delta(d, x);
  printf("delta of x and y is: %f\n", d);
  d = delta(d,d);
  printf("delta of x and y is: %f\n", d);
  return 0;
}
```

יודפס 2.3

יודפס 1.1

יודפס 0

נשים לב שהפרמטרים המועברים לפונקציה הם מטיפוס מתאים!

האם הם חייבים להיות float?

קריאה לפונקציה

- כאשר פונקציה נקראת, תחילה מחושבים ערכי הפרמטרים שהועברו אליה – אשר יכולים להיות גם ביטוי מורכב.
- הפרמטרים המועברים לפונקציה יומרו לטיפוס המתאים – בדוגמא הקודמת, פונקציה עם פרמטר מסוג float, יכולה להיקרא עם פרמטר int שיומר באופן המוכר כפי שלמדנו.
- סדר חישוב הפרמטר הוא תלוי מהדר, לכן לא ניתן להניח שהם מחושבים בסדר מסוים.
- ניתן, אך לא הכרחי, להשתמש בערך המוחזר של הפונקציה (אם היא אכן מחזירה ערך).

עוד על פרמטרים

- פרמטרים הם משתנים המוגדרים בכותרת הפונקציה ומשתחררים בסיום ריצת הפונקציה.
- פרמטרים ב – C מועברים by value, שינוי ערך פרמטר בתוך פונקציה לא ישפיע על ערכי משתנים מחוץ לפונקציה. הם בעצם עותקים של הערכים שחושבו בקריאה לפונקציה.

תרגילים

- תרגיל כתבו פונקציה המקבלת שני מספרים שלמים ומחזירה את המקסימום שלהם.

תרגיל 1 - פתרון

נשים לב לחתימת הפונקציה: שני פרמטרים מטיפוס int וערך חזרה מטיפוס int:

```
int max2(int x, int y){
  if (x > y){
      return x;
  }
  else{
      return y;
  }
}
```

יכולנו גם לכתוב return y ישר אחרי ה-if, ללא else.

תרגילים

- תרגיל 2: כתבו פונקציה המקבלת 3 מספרים שלמים ומחזירה את המקסימום ביניהם.

תרגיל 2 - פתרון

אפשר לכתוב פונקציה באותו הסגנון כמו תרגיל 1:

```
int max3(int x, int y, int z){
  if (x > y){
      if (x > z) return x;
      else return z;
  }
  \\ x is not bigger than y
  else{
      if (y > z) return y;
      else return z;
  }
}
```

```
int max2(int x, int y){
  if (x > y){
      return x;
  }
  else{
      return y;
  }
}
```

– אולי נשתמש בפונקציה הקודמת?

תרגיל 2 - פתרון

הבחנה: אילו ידענו מה המקסימום בין x ו-y, רק היינו צריכים להשוות אותו ל-z.

הערך של max2(x,y) הוא בדיוק המקסימום המדובר:

```
int max3(int x, int y, int z){
  return max2(max2(x,y),z);
}
```

נשים לב להתאמה בין טיפוס הערך המוחזר של max2 לערך החזרה של max3

נשים לב להתאמה בין טיפוס הפרמטר ש-max2 מצפה לקבל לטיפוס הערך המוחזר ע"י max2

– באופן כללי נעדיף להשתמש בפונקציות קיימות על מנת לכתוב קוד חדש.

תרגילים

- תרגיל 3: כתבו פונקציה בשם isPrime שמקבלת int כפרמטר ומחזירה true אם הפרמטר ראשוני ו- false אחרת.

תרגיל 3 – פתרון

```
bool is_prime(int num) {
    \\ if divisible by 2, only 2 is prime:
    if (num%2 == 0) ???;
    \\ otherwise see if divisible by anything
    int i=3;
    while (i*i <= num){
        if (num%i == 0) {
            ???
        }
        i++;
    }
    return true;
}
```

תרגיל 3 – פתרון

```
bool is_prime(int num) {
    \\ if divisible by 2, only 2 is prime:
    if (num%2 == 0) return num == 2;
    \\ otherwise see if divisible by anything
   int i=3;
    while (i*i <= num){
        if (num%i == 0) {
            return false;
        }
       i++;
    }
    return true;
}
```

בדיקת ראשוניות – דוגמה לשימוש

ניזכר בתרגיל בו מקבלים מספר 1-10 ומדפיסים האם הוא ראשוני, או זוגי או אי-זוגי לא ראשוני ונשנה אותו קצת.

- תרגיל 4: כתבו פונקציה בשם printProperty שפותרת את התרגיל עבור כל מספר שלם (שימו לב שהיא לא צריכה להחזיר כלום), תוך שימוש בפונקציה isPrime:

תרגיל 4 - פתרון

```
void print_property(int num){
   if (is_prime(num)){
         printf("Number is prime\n");
   }
   else if (num % 2 == 0){
         printf("Number is even\n");
   }
   else printf("Number is odd\n");
   return;
}
```

ה-return לא הכרחי

טיפוס ערך החזרה הוא void כצפוי

תרגילים

- תרגיל 5: מה לא בסדר במימושים הללו?

```
int min(int a, int b) {
  if (a > b)
     return b;
}
```

```
void print_value(int m) {
  printf("Value=%d\n", m);
  return m;
}
```

תרגילים

תרגיל 6: אילו מהקריאות לא חוקיות?

חתימות של הפונקציות:

```
int  gcd(int n, int m);
void print_value(int num);
int  dist(float, float);
char get_letter(char c);
```

```
j = gcd(j, j);
```

```
result = print_value(i+1);
```

```
dist(2.2, 1.5);
```

```
printf("input: %c\n", get_letter(k, l));
```

```
get_letter;
```

הצהרת פונקציות – בעיה

- הקומפיילר קורא את הקוד מההתחלה לסוף. כשהוא מגיע לנקודה בה יש קריאה לפונקציה, על הפונקציה הזאת להיות כבר מוגדרת:

```
int f(int x) {
  return g(x) + 1;
}

int g(x) {
  return x * -1;
}
```

- `g()` הוגדרה אחרי `f()` למרות שנעשה בה שימוש ב- `f()`
- נקבל שגיאה האומרת ש- `g()` אינה מוכרת בקריאה מ- `f()`

```
error: implicit declaration of function 'g'
```

הצהרת פונקציות – פתרון א'

- נהפוך את סדר כתיבת הפונקציות
  - הפונקציה הנקראת תיכתב לפני הפונקציה הקוראת

```
int g(int x) {
  return x * -1;
}

int f(int x) {
  return g(x) + 1;
}
```

- האם הפתרון יעבוד תמיד?

הצהרת פונקציות – בעיה

- לא ניתן ליישם את הפתרון עבור פונקציות עם קריאה מעגלית

```
int g(int x) {
  if (x <= 0)
    x = f(x);
  return x * -1;
}

int f(int x) {
  return g(x) + 1;
}
```

הצהרת פונקציות – פתרון ב'

- נצהיר על הפונקציות לפני מימושן.

חייבים ";" בסוף השורה

אין צורך לרשום את שמות המשתנים.

```
int g(int x);
int f(int);

int g(int x) {
  if (x <= 0)
    x = f(x);
  return x * -1;
}

int f(int x) {
  return g(x) + 1;
}
```

- ההצהרה מדווחת לקומפיילר על קיומן של הפונקציות לפני המימוש.
- לעיתים אף מייצאים את ההצהרות לקובץ header נפרד