מבוא לתכנות מערכות

הרצאה 2

ערך המוחזר מ-scanf, תנאים , לולאות, פונקציות, תחומי הכרה (scope) של משתנים

scanf () – הערך המוחזר

scanf () מחזירה ערך: מספר הארגומנטים שנקלטו!

```
#include <stdio.h>

int main()
{
   int  a, b;
   b = scanf ("%d", &a);
   if (b!=1)
      return 1;
   printf ( "a=%d", a);
   return 0;
}
```

אם הקלט התקבל בצורה תקינה,
b אמור לקבל ערך 1 כיוון שיש רק
קלט אחד והוא למשתנה a

scanf () – הערך המוחזר (המשך)

- צורה נוספת לאותה תכנית:

```
#include <stdio.h>

int main()
{
  int  a;
  if (scanf ("%d", &a) != 1)
     return 1;
  printf("a=%d", a);
  return 0;
}
```

תנאים

- באופן רגיל ביצוע של תכנית הוא סדרתי: הוראות מתבצעות בזו אחר זו.
- הוראות בקרה מאפשרות חריגה מהביצוע הסדרתי.

הוראות לבקרת הזרימה (control flow)

לולאות

הוראות תנאי

אופרטורים לוגיים

AND, OR, NOT

p	q	p || q
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

טבלת אמת לפעולה OR

p	q	p&&q
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

טבלת אמת לפעולה AND

p	!p
FALSE	TRUE
TRUE	FALSE

טבלת אמת לפעולה NOT

משפט תנאי if

שאחריו הוראה בודדת או בלוק הוראות

```
if (condition) statement
```

```
if (ch>='a' && ch<='z')
        printf("a lower-case letter");
```

```
if (condition)
{
    several statements
}
```

```
if ( x > y )
 {
        max=x;
        min=y ;
 }
```

הוראת if - else

```
if (n > 0)
{
    if (a > b)
    {
        z=a;
        a++;
    }
    else
        z=b;
}
```

הערה חשובה:

מומלץ תמיד להשתמש בסוגריים מסולסלים בהוראות if, אלא אם כן יש רק פקודה אחת לביצוע:

- משמעות ברורה וחדה.
- הוספת הוראות לגוף (במקרה הצורך) ללא פגיעה בתקינות התחבירית.
- כלל: (!!!)

else שייך ל- if שלפניו הקרוב אליו ביותר.

הסתעפויות מרובות סעיפים

הוראת if-else מאפשרת להבחין בין שני מצבים. לעיתים רבות יש להבחין בין מספר מצבים שונים:

```
if   ( expression)
        statement
else   if   ( expression)
        statement
else   if   ( expression)
        statement
else
        statement
```

- התנאים נבדקים לפי הסדר.
- אם אחד התנאים נכון, ההוראה הצמודה לו מתבצעת והשרשרת נקטעת.
- ה- else האחרון הוא ברירת מחדל – הוא מציין את המקרה שבו אף אחד מן התנאים אינו מתקיים. אם אין מצב כזה, ניתן להשמיטו (או להשתמש בו על מנת לאתר טעויות).

דוגמא - מחשבון פשוט

משתנה op הוא מסוג char.

```
if (op == ‘+’)
     result = x + y;
else
    if (op == ‘–’)
            result = x – y;
    else
            if (op == ‘*’)
                 result = x * y;
            else
                if (op == ‘/’)
                      result = x / y;
                else
                        printf(“Error, unknown operator.”);
```

שגיאה נפוצה

שימוש בסימן = במקום == כאשר משווים בין ערכים.

הקומפיילר לא יתריע. זו "שגיאה לוגית" ולא "תחבירית". מדוע?

```
if (x=5) …
```

ב- C יש משמעות לביטוי זה כי למשפט השמה יש ערך!

מתבצעות שתי פקודות הבאות אחת אחרי השנייה:

```
x=5;
if(x);      /*if(x!=0)-זה שקול ל */
```

עוד שגיאה נפוצה

- חשבו את הביטוי:
- `2 < X < 15`
- נָרְאֶה כי ערך הביטוי אינו תלוי בערכו של X ,
- נניח X=3
- $(2 < 3) < 15 \equiv \text{TRUE} < 15 \equiv 1 < 15 \equiv \text{TRUE} \equiv 1$
- נניח כי X= 1
- $(2 < 1) < 15 \equiv \text{FALSE} < 15 \equiv 0 < 15 \equiv 1$

ביטוי כזה נכון במתמטיקה אך לא ב- C.

הביטוי המתאים ב- C הוא: **x>2 && x<15**

(הערה: כמו במקרים רבים אחרים, הביטוי השגוי ניתן לחישוב ב- C... ערכו יהיה 1 (אמת) עבור כל ערך של x .)

האופרטור :?

משמש לקיצור משפט if במקרים פשוטים.

<u>מבנה המשפט:</u> **(condition) ? expr1 : expr2**

שקול ל-:

```
if (condition) {
    value = expr1;
}  else {
    value = expr2;}
```

- בעיה: קלוט שני מספרים שלמים, הדפס את שם המשתנה הגדול.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
   int a, b;
   printf ("please insert the two integers: ");
   scanf ("%d %d", &a, &b);
  printf ("%c = %d is the biggest", (a>b)?'a':'b', (a>b)?a:b );
  return 0;    }
```

הפקודה break

- משמעות break: "הפסק מייד את בצוע משפט הבקרה שבו הפקודה כתובה".
- break ניתן להפעלה במשפטי הבקרה for , while, do-while ו – switch.
- אם משפט הבקרה מקונן בתוך משפט אחר, אז break מסיים את בצוע <u>משפט הבקרה הפנימי</u>, ומעביר את בקרת התכנית אל משפט הבקרה החיצוני.

הוראת switch

תחביר:

```
switch (expression) /*The expression can be integer expression or a character expression*/
{
    case const-expr:    statements
    case const-expr:    statements
    :
    case const-expr:    statements
    default:            statements
}
```

- מילים שמורות
- חלק מן התחביר
- statements: סדרה של אפס או יותר הוראות (אין צורך בהוראה מורכבת כדי לפרט מספר הוראות בסעיף אחד!).
- const-expr: ביטוי שערכו צריך להיות קבוע, כלומר ידוע כבר בזמן קומפילציה. אסור ששני const-expr יהיו שווים.
- השורה האחרונה, שהתוית שלה היא המילה השמורה default, היא אופציונאלית.

הוראת switch

משמעות:

```
switch (expression)
{
        case const-expr:        statements
        case const-expr:        statements
        :
        case const-expr:        statements
        default:                statements
}
```

- בזמן ריצה, expression משוערך וערכו מושווה לכל אחד מה- const-expr השונים, על פי סדר כתיבתם.
- ברגע שנמצאת התאמה (expreession=const-expr) הביצוע מתחיל בהוראות הצמודות ל- case המתאים.
- הביצוע ממשיך ("נופל") לפקודות של ה- case-ים הבאים, אלא אם כן ננקט אמצעי מפורש למנוע זאת (למשל באמצעות הוראת break הגורמת לביצוע לצאת מה- switch).
- אם אף case אינו תופס, יתבצעו ההוראות הצמודות לתווית default אם יש כזו. אם אין תווית כזו, לא תתבצע כל הוראה והביצוע ימשיך אל ההוראה שלאחר ה- switch.

הוראת switch - דוגמא

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
        char ot;
        int a, b, c;
        printf (" Please insert a char and three numbers: ");
        scanf ("%c %d %d %d", &ot, &a, &b, &c);
        switch (ot)
       {
            case '+': printf (" sum is %d", a + b + c);
                        break;
            case '*': printf (" product is %d", a * b * c);
                        break;
            case 'a': printf (" average is %f", (a + b + c) / 3.0 );
                        break;
            default:
                        printf (" error, illegal operation\n" );
        }
        return 0;
}
```

הוראת switch - דוגמא (2)

```
/* A program for demonstrating switches */
/* This program reads a character, interpreted as a Y/N answer, then prints the answer  */

#include <stdio.h>

int main()
{
        char c;
        printf("Yes or No?\n");
        scanf("%c", &c);
        switch (c)
        {
                        case 'y': case 'Y':
                                printf("Yes\n");
                                break;
                        case 'n': case 'N':
                                printf("No\n");
                                break;
                        default:
                                printf("Invalid answer.\n");
                                break;
        }
                return 0;
 }
```

לולאות

הרעיון - ביצוע של הוראה אחת, או סדרה של הוראות, מספר כלשהוא של פעמים:

- מספר הפעמים ידוע מראש (למשל – סיכום חמישה מספרים הנתונים בקלט).

או

- מספר לא ידוע מראש של פעמים. התנאי לסיום הביצוע נקבע באופן דינאמי בזמן ריצת התכנית.

לולאות while

תחביר:

```
while  (expression)
{
        statement1;
        statement2;
      ……..
}
```

- while – מילה שמורה
- הסוגריים הם חלק מן התחביר
- expression – ביטוי
- statement – הוראה אחת

לולאות while

משמעות:

```
while  (expression)
        statement
```

1. expression משוערך.
2. אמת ← ההוראה statement מתבצעת והביצוע חוזר אל expression.

שקר ← הבקרה מועברת אל ההוראה הבאה לאחר ה- while.

לולאות while - דוגמא

```
/* This program computes n!, the factorial of n */
#include <stdio.h>
int main()
{
        int n,i;
        long int result=1;
        printf("Enter a natural number: ");
        if (scanf("%d", &n) != 1) {
                printf("Input error.\n");
                return 1;
        }
        else if (n<0) {
                printf("Factorial is undefined for negative integers.\n");
                return -1;
        }
        else {
                i=1;
                while (i<=n) {
                        result = result*i;
                        i++;
                }
                printf("%d!=%ld\n", n, result);
        }
        return 0;
}
```

n! הוא מספר שלם לכל n, אבל ערכו גדל במהירות (מקום אחסון מספיק)

בקרת קלט:

- האם המספר שלם?
- האם המספר חיובי?

לולאות while

```
/* This program computes n!, the factorial of n */
#include <stdio.h>
int main()
{
        int n,i;
        long int result=1;
        printf("Enter a natural number: ");
         if (scanf("%d", &n) != 1) {
                printf("Input error.\n");
                return 1;
        }
        else if (n<0) {
                printf("Factorial is undefined for negative integers.\n");
                return -1;
        }
        else {
            i=1;
            while (i<=n) {
                result *= i;
                i++;
            }
            printf("%d!=%ld\n", n, result);
        }
        return 0;
}
```

1. אתחול משתנה המשמש כמונה(פעולה חד-פעמית)
2. תנאי הבודק את ערכו של המשתנה
3. גוף הלולאה
4. עדכון המשתנה

לולאה אינסופית

```
while (1)
    printf ("Matam is fun!\n");
```

- מאחר ותנאי הלולאה הוא תמיד אמת כי 1 שונה מ-0,גוף הלולאה יתבצע תמיד וזו תהיה לולאה אינסופית. ניתן לצסיים את הלולאה הזאת ע"י שימוש ב-break.

לולאות do while
לקריאה עצמית של סטודנטים

```
#include <stdio.h>
int main()
{
        int num, max = 0;
        do
        {
                printf ("Insert a positive number: ");
                scanf ("%d",  &num);
                max =  (max >=  num)? max : num;
        }
        while (num);                    /*while num !=0*/
        printf ("The maximum value is %d \n", max );
}
```

לולאות for

(*) אתחול – ביטוי השמה

(*) בדיקת תנאי – ביטוי יחס (מחזיר ערך לוגי)

(*) קידום – ביטוי השמה

```
for  (expr1 ; expr2 ; expr3)
        statement
```

- מבחינה תחבירית, כל אחד משלושת הביטויים יכול להיות מושמט, אך סימני ה- ; הכרחיים.
- מבחינה סמנטית, הלולאה תתבצע כל עוד expr2 משוערך כאמת (כלומר, 0≠). אם expr2 חסר, הוא מתפרש כאמת תמיד.

לולאות for - דוגמא

```
/* This program computes n!, the factorial of n */
#include <stdio.h>
int main()
{
        int n,i;
        int result=1;
        printf("Enter a natural number: ");
         if (scanf("%d", &n) != 1) {
                printf("Input error.\n");
                return 1;
        }
        else if (n<0) {
                printf("Factorial is undefined for negative integers.\n");
                return -1;
        }
        else {
                 for (i=1; i<=n ; i++) {
                      result = result*i;
                }
                printf("%d!=%ld \n", n, result);
       }
        return 0;
}
```

אפשר גם לאתחל את המשתנה result בתחילת הלולאה, כפי שנראה בשקף הבא

לולאות for - דוגמא

```
/* This program computes n!, the factorial of n */
#include <stdio.h>
int main()
{
        int n,i;
        long int result;
        printf("Enter a natural number: ");
         if (scanf("%d", &n) != 1) {
                printf("Input error.\n");
                return 1;
        }
        else if (n<0) {
                printf("Factorial is undefined for negative integers.\n");
                return -1;
        }
        else {
                for (i=1 , result=1 ; i<=n ; i++) {
                        result =result* i;
                }
                printf("%d!=%ld\n", n, result);
       }
        return 0;
}
```

לולאות for – (המשך)

סדר פעולות בביצוע לולאת for:

1. בצע את expr1 - מצב התחלתי של לולאה,
2. חשב את expr2 - תנאי להמשך ביצוע הלולאה,
3. **אם** התנאי מתקיים (ערכו הוא TRUE) אזי:
   a) בצע את statement – גוף הלולאה, משפט או {}.
   b) בצע את expr3,
   c) חזור לסעיף 2,

   אחרת:
4. סיים את הלולאה, עבור אל הפקודה הבאה שמעבר ללולאה.

לולאות for

- תחביר המשפט:

- for (exp1; exp2; exp3) statement;

❑ התחביר חובה

- וגם שקול לקוד הבא :

- ```
  exp1;
  for (;exp2;)
  {
  	statement;
  	exp3;
  }
  ```

- קוד זה שקול לקוד הבא :

- ```
  exp1;
  while (exp2)
  {
  	statement;
  	exp3;
  }
  ```

פונקציות – יתרונות

- פישוט תהליך פיתוח הקוד
- קוד קריא יותר
- קל לבצע שינויים
- חסכון בכתיבה (קטע קוד המבוצע מספר פעמים)
- מחזור קוד

בדר"כ תכניות ב-C מורכבות ממספר גדול של פונקציות קטנות ולא ממספר קטן של פונקציות גדולות!

פונקציות – תחביר

1. הצהרה declaration

מבהירה לקומפיילר שבכוונת המתכנת להשתמש בפונקציה מסוימת.

2. הגדרה definition

פירוט המימוש של הפונקציה - שם הפונקציה, המספר והטיפוס של הפרמטרים שהפונקציה פועלת עליהם וטיפוס הערך המוחזר מהפונקציה.

3. שימוש/קריאה call

פונקציות

- הגדרת פונקציה:

```
<type> <name>(parameters list)
{
      local variables…
      body
}
```

- <type> טיפוס הערך המוחזר על ידי הפונקציה (void אם לא מחזירה דבר)
- <name> שם הפונקציה,
- (parameters list) רשימת **הפרמטרים** שפונקציה מקבלת,
- body – גוף הפונקציה. בלוק הפקודות לביצוע.

קריאה לפונקציה

- מחושבים הביטויים הנמצאים ברשימת הארגומנטים המופיעים בקריאה לפונקציה לאחר החישוב – ערך כל ביטוי מוכנס למשתנה המתאים – אלו שהגדרנו בכותרת הפונקציה
- זוהי העברת משתנים "call by value", כלומר מועבר רק הערך של המשתנים (ולא כתובתם!)
- מוקצים המשתנים המוגדרים בגוף הפונקציה ומתבצע קוד הפונקציה.

דוגמא לקריאה

```
#include <stdio.h>

int max(int a, int b);

int main()
{
    int x, y;
    printf("insert 2 numbers: ");
    scanf ("%d%d", &x, &y);
    printf("max = %d\n", max(x+12, y/3));
    return 0;
}

int max(int a, int b)
{
    return((a>b)? a:b);
}
```

פונקציות - דוגמא

```
/* This program computes m to the power of n */
/* Assumptions: m is an integer; n is a positive integer */

#include <stdio.h>

int power(int, int);

/* test the power function */
int main()
{
        int i;
        for (i=0; i<10; i++) {
                printf("%d %d %d\n", i, power(2,i), power(3,i));
        }
        return 0;
}

/* power: raise base to n-th power; n>=0 */
int power(int base, int n)
{
        int i, p=1;
        for (i=1; i<=n; i++) {
                p = p*base;
        }
        return p;
}
```

הצהרה על הפונקציה power
(function prototype)

קריאה לפונקציה power

הגדרת הפונקציה power:

- כותרת
- גוף

פונקציות - הצהרה

הצהרה (declaration):

- טיפוס הערך שהפונקציה מחזירה (פונקציות ב- C מחזירות ערך אחד ויחיד.
- שם הפונקציה.
- הסוגריים - הכרחיים.
- בין הסוגריים מפורטים הפרמטרים של הפונקציה. לכל פרמטר, שם וטיפוס.
- שמות הפרמטרים ניתנים להשמטה בהצהרה, כך שההצהרה שלהלן שקולה:

```
int power (int,int);
```

```
/* This program computes m to the power of n */
/* Assumptions: m is an integer; n is a positive integer */

#include <stdio.h>

int power(int, int);

/* test the power function */
int main()
{
	int i;
	for (i=0; i<10; i++) {
		printf("%d %d %d\n", i, power(2,i), power(3,i));
	}
	return 0;
}

/* power: raise base to n-th power; n>=0 */
int power(int base, int n)
{
	int i, p=1;
	for (i=1; i<=n; i++) {
		p = p*base;
	}
	return p;
}
```

חזרה מהפונקציה

- ניתן להחזיר ערך לפונקציה הקוראת על ידי return ,
- עם ההגעה אל משפט ה- return או לסוף הפונקציה:
  - משוחרר הזיכרון שהוקצה למשתנים שהוגדרו בתוך הפונקציה, כלומר, הם נמחקים!
  - מייד חוזרים לבצע את המשך התוכנית (הפונקציה הקוראת) מהמקום בו היא הופסקה.
- במידת הצורך מתבצעת המרת טיפוסים אוטומטית:
  הערך המוחזר מומר בהתאם לטיפוס ההחזרה המצוין בהגדרת הפונקציה ,
- כאשר אין צורך שהפונקציה תחזיר ערך, נכתוב את המילה void, כטיפוס הערך המוחזר.
- ניתן גם לכתוב return;

(ללא expr) ואזי הפונקציה לא תחזיר ערך.

אורך חיים וטווח הכרה

- תוכנה ב- c כתובה בבלוקים,
- משתנים לוקליים (מקומיים) אפשר להגדיר רק בתחילת כל בלוק (בלוק זה מה שמופיע בין הסוגריים {}. למשל,בתחילת פונקציה.
- המשתנה מוכר בקטע התוכנה שבו הוגדר:

```
…
{
        int b;
        b = 2;
}
…
```

בלוק פנימי,
כולל הגדרת b,
ושימוש בו.

אורך חיים וטווח הכרה (המשך)

- כאשר יש בלוק תוכנה חיצוני, המכיל בלוק פנימי,כל המזהים של הבלוק החיצוני מוכרים בפנימי,
- אבל, המזהים של הבלוק הפנימי אינם מוכרים בחיצוני:

```
int a …
{
        int b;
        b = b + a;
}
…
```

בלוק חיצוני.
b אינו מוכר כאן ,

בלוק פנימי,
כולל הגדרת b,
ושימוש בו.
גם a מוכר כאן.

אורך חיים וטווח הכרה

- וְמָה אִם... השם כבר מופיע בבלוק אחר?
- שם בבלוק פנימי הזהה לשם שנמצא בבלוק חיצוני, "ממסך" את המזהה של הבלוק החיצוני:

```
int a =1;
{
        int a = 2, b;
        b = a;        /* b ←2 */
}
…
```

בלוק חיצוני.
כולל הגדרת a.

בלוק פנימי.
כולל הגדרת a.
זהו a **שונה** מאשר
בבלוק החיצוני!

משתנים לוקליים– סיכום ביניים

- בלוק: קטע קוד שמתחיל ב-{ ונגמר ב-}.
- פונקציה, למשל, היא בלוק.
- טווח ההכרה של משתנה מקומי הוא בבלוק שבו הוגדר.
- המשתנה יהיה מוכר גם בבלוקים המקוננים בבלוק זה,
  בתנאי – שבבלוקים המקוננים לא הוגדר משתנה עם אותו שם.
- אין הכרה במשתנים מבלוקים מקבילים, או פנימיים.

פונקציות ומשתנים לוקליים

- המשתנים של הפונקציה הם לוקליים,
- אורך חייהם הוא מרגע הקריאה ועד היציאה מהפונקציה . ביציאה מהפונקציה משתחרר הזכרון של משתנים לוקליים.
- אין הכרה במשתנים שנמצאים בפונקציות אחרות,
ופונקציות אחרות (כולל main) אינן מכירות משתנים לוקליים של פונקציה אחרת.

משתנים גלובליים

- משתנים גלובליים הינם משתנים המוצהרים מחוץ לבלוקים ופונקציות,
- אורך חייהם הוא כאורך חיי התוכנית,
- טווח הכרתם הוא מהגדרתם ו"מטה" כולל כל הבלוקים,
- גם כאן, אם יש משתנה מקומי בשם זהה לגלובלי, הוא "יִמָּסֵך" את הגישה למשתנה הגלובלי ,
- עבודה עם משתנים גלובליים מנוגדת לרעיון של חלוקה למשימות עצמאיות, מקשה על פיתוח התוכנית, ועלולה להוות פרצת אבטחה.
  דמיינו קוד הנכתב על ידי 30 תכנתים במשותף!
- בקורס הזה לסטודנטים אסור שימוש במשתנים גלובליים.

משתנים גלובליים (המשך)

```
int x;
int main()
{
    int a…..
    {
        int b;
        b = b + a;
    }
}
```

global variable,

inner block,

yet, inner block ,

- שימו לב:
  משתנה המוגדר מחוץ לבלוק מסוים, נראה מתוך הבלוק כמו משתנה גלובלי.

משתנים סטטיים

- זוהי תכונה נוספת שניתן לייחס למשתנים, המיוחד שבה הוא ***אורך*** חיי המשתנים.
- ניתן להגדיר משתנים כסטטיים בראשית בלוק.
- משתנים סטטיים, "נולדים" עם הכניסה לבלוק, כמו משתנים לוקליים רגילים,
  אך אינם "מתים" עם היציאה ממנו – אלא מתקיימים עד סוף התוכנית, ותוך כדי כך ניתן לעדכן את ערכם,כלומר, שורת ההגדרה שלהם מתבצעת רק פעם אחת .
- משתנים סטטיים מקבלים איתחול אוטומטי ל-0 (אם אין איתחול אחר), בניגוד לשאר הסוגים.
- ניתן לגשת למשתנה סטטי בבלוק שהוא הוגדר בו (ובבלוקים הפנימיים), בדיוק כמו למשתנה לוקאלי.
- צורת ההגדרה:

```
static <type> <name>;
```

משתנים סטטיים – דוגמא

```
#include <stdio.h>

void f_x();

int  main()
{
    int i;
    for ( i=0; i<10; i ++)
      f_x();
    return 0;
}

void f_x()
{
    static int cnt = 1;
    printf("%d", cnt);
    cnt = cnt +1;
}
```

- מה יהיה הפלט?

1 2 3 4 5 6 7 8 9 10