

קורס תכנות

שיעור שני: שימוש במשתנים,
בקרת זרימה

נושאי השיעור היום

- **משתנים (variables)**
 - טיפוסים משתנים בשפת C
 - הגדרת משתנים
 - השמה למשתנים
 - פעולות על משתנים
 - קליטת ערכים מהמשתמש
 - הדפסה משתנים
- **בקרת זרימה**
 - משפטי if

חישוב מספר הדקות ביממה

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

```
int main()
```

```
{
```

הצהרה

```
int hours, minutes, total;
```

```
hours = 24;
```

השמה

```
minutes = 60;
```

```
total = hours * minutes;
```

ביטוי אריתמטי

```
printf("Minutes in a day: %d\n", total);
```

ביטוי (ערך לפונקציה)

```
return 0;
```

```
}
```

בכל שורות הקוד
המסומנות יש פעולה
כלשהי המבוצעת על או
בעזרת משתנים

מה זה משתנה?

- **מקום בזיכרון ש:**
 - שייך לתוכנית
 - הוגדר מה טיפוס הערך מאוחסן בו
 - קיבל שם (שבעזרתו ניגש לזיכרון)
 - ניתן לקרוא/לכתוב אליו ערכים
- משתנים (**variables**) הם האמצעי לטיפול בנתונים בתוכנית

הגדרת משתנים

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

```
int main()
```

```
{
```

```
    int hours, minutes, total;
```

```
    hours = 24;
```

```
    minutes = 60;
```

```
    total = hours * minutes;
```

```
    printf("Minutes in a day: %d\n", total);
```

```
    return 0;
```

```
}
```

הגדרת משתנים:

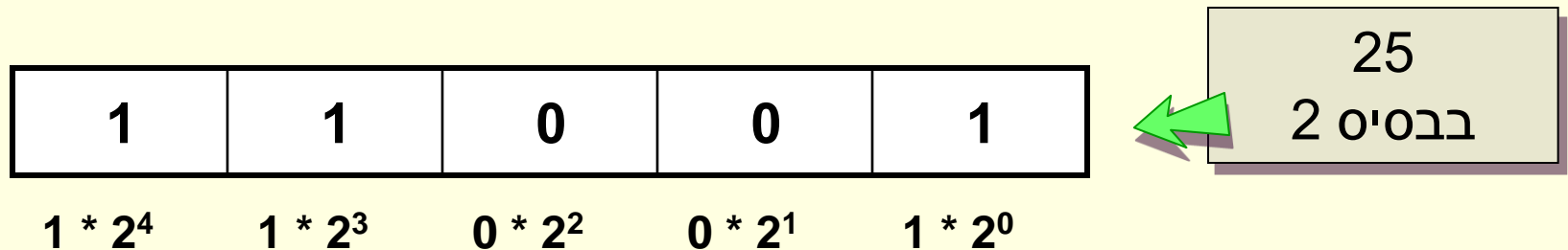
- הגדרת **טיפוס ושם**
- אתחול המשתנה (לא חובה)
- מוגדרים בתחילת block

:block

- אוסף המשפטים המוגדר ע"י זוג { }

ייצוג ערכי משתנים במחשב

- מספרים במחשב מיצגים בשיטה בינארית (בסיס 2)
- כל ביט (bit) בזיכרון מייצג ספרה אחת
- ככל שנשתמש ביותר ביטים נוכל לייצג טווח ערכים גדול יותר
- בעזרת n ביטים ניתן לייצג 2^n ערכים
- byte הוא רצף של שמונה ביטים

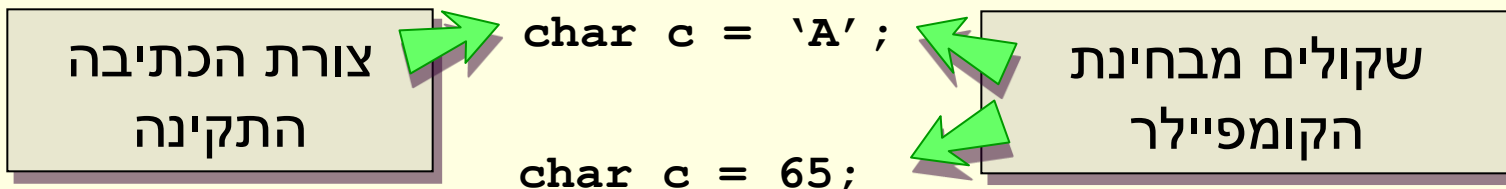


טיפוסים

- מייצג סוג מסוים של מידע (מספר שלם, תו, מספר ממשי)
- **int** (קיצור של integer):
 - מספר שלם
 - גודל: לא מוגדר, 2 או 4 בתים (בד"כ 4)
 - טווח ערכים: -2^{31} ל 2^{31}
- **float**:
 - מספר ממשי
 - גודל: 4 בתים
- **long**: שלם, כפול בגודלו מ int
- **double**: ממשי, כפול בגודלו מ float, מאפשר רמת דיוק גבוהה יותר

הטיפוס char

- מייצג תווים:
 - כל תו נשמר בתא (byte) בודד
 - המחשב מבין רק מספרים ← גם תווים נשמרים בזיכרון כמספרים
- טבלת ASCII:
 - המרה של 256 תווים שונים למספר סידורי 0-255
 - האותיות A-Z נמצאות בטבלה לפי הסדר במקומות 65-90
 - האותיות a-z נמצאות בטבלה לפי הסדר במקומות 97-122
 - הספרות נמצאות בטבלה לפי הסדר במקומות 48-57



טבלת ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTable.com

למה צריך טיפוסים משתנים?

- חסכון במקום:
 - טיפוסים מסוימים דורשים פחות מקום בזיכרון מאחרים
- מהירות:
 - המחשב יטפל אחרת בטיפוסים משתנים שונים
 - לדוגמא כפל שלמים מבוצע מהר יותר מכפל ממשיים
- איתור שגיאות:
 - בכתיבה ובזמן קומפילציה
- אין ציון של הטיפוס או מקום ההתחלה של המשתנה בזיכרון
- הקומפיילר דואג שהתוכנית תתייחס למשתנים בהתאם לסוג שהוגדר

הגדרת משתנים

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

```
int main()
```

```
{
```

```
int hours, minutes, total;
```

```
hours = 24;
```

```
minutes = 60;
```

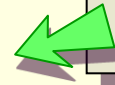
```
total = hours * minutes;
```

```
printf("Minutes in a day: %d\n", total);
```

```
return 0;
```

```
}
```

כאן אנו מגדירים שלושה
משתנים מסוג `int`



שמות משתנים

- שם משתנה:
 - יכול לכלול:
 - אותיות אנגליות (abc...)
 - ספרות (012...)
 - וקו תחתון (_)
 - התו הראשון בשם **לא יכול** להיות סיפרה
 - יש הבדל בין אות גדולה לאות קטנה (**case-sensitive**)
 - כלומר a ו- A הם שמות שונים

counter, new_sum, total1

• חוקי: ←

percent%, 2counter, new-sum

• לא חוקי: ←

הגדרת משתנים

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

```
int main()
```

```
{
```

```
    int hours, minutes, total;
```

```
    hours = 24;
```

```
    minutes = 60;
```

```
    total = hours * minutes;
```

```
    printf("Minutes in a day: %d\n", total);
```

```
    return 0;
```

```
}
```

רצוי לתת למשתנים שמות
בעלי משמעות



הגדרת משתנים

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

```
int main()
```

```
{
```

```
    int hours, minutes, total;
```

```
    hours = 24;
```

```
    minutes = 60;
```

```
    total = hours * minutes;
```

```
    printf("Minutes in a day: %d\n", total);
```

```
    return 0;
```

```
}
```



לאחר שלב זה המקומות בזכרון מוקצים אבל לא ידוע מהם הערכים שנמצאים שם

total minutes hours

???

???

???

זיכרון

השמה

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

```
int main()
```

```
{
```

```
    int hours, minutes, total;
```

```
    hours = 24;
```

```
    minutes = 60;
```


```
    total = hours * minutes;
```

```
    printf("Minutes in a day: %d\n", total);
```

```
    return 0;
```

```
}
```

רק לאחר ההשמה מופיעים
בזיכרון הערכים שאנו רוצים
שיופיעו שם



total minutes hours

1440

60

24

זיכרון

איתחול משתנים

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

```
int main()
```

```
{
```

```
int hours=0, minutes=0, total=0;
```

```
hours = 24;
```

```
minutes = 60;
```

```
total = hours * minutes;
```

```
printf("Minutes in a day: %d\n", total);
```

```
return 0;
```

```
}
```

השמת ערך בזמן הגדרת
המשתנה נקראת איתחול

איתחול מבטיח שתמיד
יופיע בזיכרון ערך "סביר"

total minutes hours

0	0	0
---	---	---

זיכרון

דוגמאות להגדרות משתנים

- `int i = 0;`
- `float max_percent = 0.0f;`
- `double average = 0.0;`
- `char character = 'A';`
- `int sum = 0;`
- `int a = 0, b, c = 100;`

רצוי לאתחל משתנים
רצוי לתת להם שמות משמעותיים



- במצגות מפאת חוסר מקום:
- לא תמיד נאתחל משתנים
 - לפעמים נקצר בשמות משתנים

ביטויים

- ביטוי הוא מבנה בעל ערך
- השפה מגדירה ביטויים ראשוניים ודרכים לבנות ביטויים מורכבים מאותם ביטויים ראשוניים
- ביטויים ראשוניים
 - קבוע – 10, 'A', 0.7
 - משתנה
 - מחרוזת – "Hello World"

אופרטורים

- ניתן ליצור ביטויים מורכבים בעזרת הפעלה של אופרטורים על ביטויים פשוטים יותר
- $x + 1$
- אופרטורים שימושיים:
 - השמה =
 - סוגריים ()
 - אריתמטיים: +, -, *, /, % (מודולו)
 - יחס: <, >, <=, >=, ==, !=
 - לוגיים: (AND) &&, (OR) ||
 - המרה (casting)

פעולות – נקודות לתשומת-לב

- בחלוקת מספרים **שלמים** תתקבל המנה ללא השארית:
- $7 / 2 = 3$
- בפעולה על משתנים מסוגים שונים, המחשב ימיר את המשתנים לסוג בעל טווח/דיוק הגדול מביניהם.
- $7 / 2.0 = 3.5$
- אם נכניס למשתנה ערך גדול מהטווח המקסימאלי שלו, נקבל תוצאה שגויה (**overflow**)

המרה - Casting

- ממירים את הטיפוס של הביטוי

```
int valueOne = 1, valueTwo = 2;  
double result = 0.0;
```

```
result = (double)valueOne / valueTwo;
```

- המשתנה valueOne הוגדר כ int, לאחר הפעלת ההמרה נקבל ביטוי חדש מטיפוס double
- הפעלת אופרטור החילוק כעת תהיה בין ביטוי שלם וביטוי ממשי והביטוי השלם יומר (באופן אוטומטי) לממשי
- הטיפוס של כל הביטוי המורכב הוא double
- והפלט יהיה: 0.5

הדפסה של ערכים - printf

- הדפסת ערך של ביטויים (ים)

```
printf(ביטויים, מחרוזת פורמט);
```

- לדוגמא: `printf("The sum is %d\n", sum);`

- על מנת להדפיס ערך של משתנה יש להשתמש בתו `%`:

- `char – %c`

- `int – %d` בבסיס 10

- `float – %f`

- `double – %lf`

- `%e` – הצגת מספר עשרוני עם מעריך (2.154e+001)

- `%g` – ההצגה הברורה יותר מבין `%e` ו-`%f`

דוגמא: כפל של שני מספרים ממשיים

```
#include <stdio.h>

int main()
{
    double a=0.0, b=0.0, c=0.0;

    a = 2.51;
    b = 2;
    c = a*b;

    printf("The product is %lf\n", c);
    return 0;
}
```

נדפיס את הערך של
המשתנה c

הפלט הוא: "The product is 5.02"

דוגמא: כפל של שני מספרים ממשיים

```
#include <stdio.h>

int main()
{
    double a=0.0, b=0.0, c=0.0;

    a = 2.51;
    b = 2;

    printf("The product is %lf\n", a * b);
    return 0;
}
```

הפלט הוא: "The product is 5.02"

דוגמא: כפל של שני מספרים ממשיים


```
#include <stdio.h>

int main()
{
    double a=0.0, b=0.0, c=0.0;

    a = 2.51;
    b = 2;
    c = a*b;

    printf("%lf * %lf = %lf\n", a, b, c);
    return 0;
}
```

נדפיס את הערך של
המשתנים a, b, c



הפלט הוא: "2.51 * 2.0 = 5.02"

קלט של ערכים מהמשתמש - scanf

- לרוב נרצה לעבוד על **קלט שמתקבל המשתמש**
- הפונקציה scanf קולטת ערך מהמשתמש לתוך משתנה

```
scanf(כתובות, מחרוזת פורמט);
```

```
scanf("%d %lf", &student_num, &average);
```

- התכנית ממתינה לקלט מהמשתמש
- אחרי הקשת הקלט על המשתמש להקיש **Enter** על המקש
- הקלט נכנס למשתנה שצוין
- סימוני טיפוס הקלט זהים לסימוני ההדפסה ב- printf
- לפני שם המשתנה יש לשים את הסימן **&** ("הכתובת של")

דוגמא: הדפסת קלט מהמשתמש

```
#include <stdio.h>

int main()
{
    int a;

    printf("Enter an integer\n");
    scanf("%d", &a);

    printf("The input is: %d\n", a);

    return 0;
}
```

שימו לב לתו '&' המופיע לפני שם המשתנה ב scanf
ולא מופיע ב printf

תרגום מפרנהייט לצלזיוס

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

```
int main()
```

```
{
```

```
double celsius = 0.0, fahrenheit = 0.0;
```

הגדרת משתנים

```
printf("Please enter a fahrenheit temperature:\n");
```

```
scanf("%lf",&fahrenheit);
```

קליטת ערכים מהמשתמש

```
celsius = 5*(fahrenheit - 32) / 9;
```

חישוב והשמת ערך

```
printf("This is equal to %lf degrees celsius\n", celsius);
```

```
return 0;
```

הדפסת הערך למסך

```
}
```

שלמים חיוביים

- לעיתים נרצה להשתמש במשתנה לייצוג מספרים חיוביים בלבד
 - מגדיל את הטווח פי 2
 - בהגדרת המשתנה נוסיף את המילה **unsigned** לפני הטיפוס
 - לטיפוסים שלמים בלבד (int, char)
- ```
unsigned int i = 3999999999;
```
- ההדפסה נעשית עם **%u**

# קבועים

- משמשים לערך שאינו משתנה במהלך התכנית
  - למשל -  $\pi$
  - חוסך חזרה על מספרים
  - מונע שינוי הערך בטעות
- מוסיפים **const** בהגדרה

```
const double pi = 3.141592654;
```

# משפטים Statements

- יחידת ביצוע שלמה
- יחידת הקוד הקטנה ביותר הניתנת לביצוע
- בדרך כלל תאופיין בתו ';' המופיע בסופה
- דוגמאות:
- הגדרת משתנים
- משפטי ביטוי – ביטוי שלאחריו ';'
- Block – משפט המתחיל ב '{' ומסתיים ב '}' ובתוכו אפס או יותר משפטים אחרים
- תכנית מחשב בנויה מסדרה של משפטים המבוצעים אחד אחרי השני

- נרצה לכתוב תכנית הקולטת שני מספרים שלמים מהמשתמש ומדפיסה את הגדול מביניהם
- נאחסן את המספרים במשתנים  $a$  ו- $b$
- כיצד נבצע הדפסה של  $a$  רק אם הוא הגדול מבין השניים?
- באופן כללי, כיצד נבצע משפטים מסוימים בתכנית רק בתנאי כלשהו?



# משפט if

- משמש לביצוע מותנה של משפט (או מספר משפטים) בתכנית

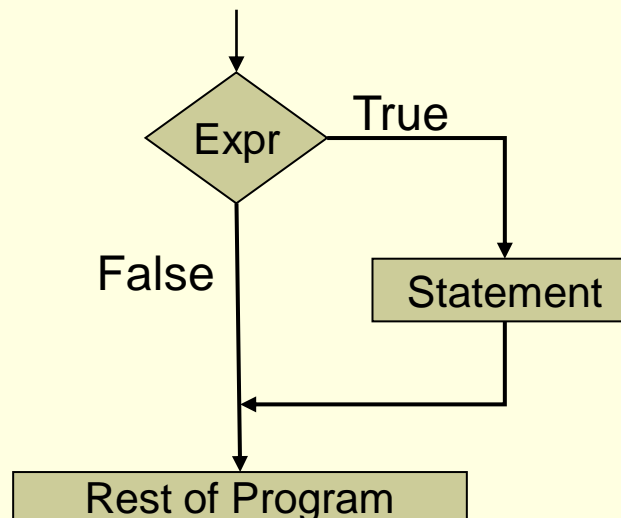
```
if (expression)
 statement
```

- תחביר:

```
if (a > b)
 printf("%d\n", a);
```

- דוגמא:

שימוש באופרטור  
היחס > כדי ליצור ביטוי  
מורכב משני הביטויים  
הראשונים a ו-b-



# הדפסת המספר המקסימאלי

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

```
int main()
```

```
{
```

```
 int a, b, max;
```

```
 scanf("%d%d", &a, &b);
```

קלוט שני מספרים שלמים  
מהמשתמש

```
 max = a;
```

```
 if (max < b)
```

```
 max = b;
```

ייתבצע רק אם התנאי מתקיים

```
 printf("The maximum is %d\n", max);
```

```
 return 0;
```

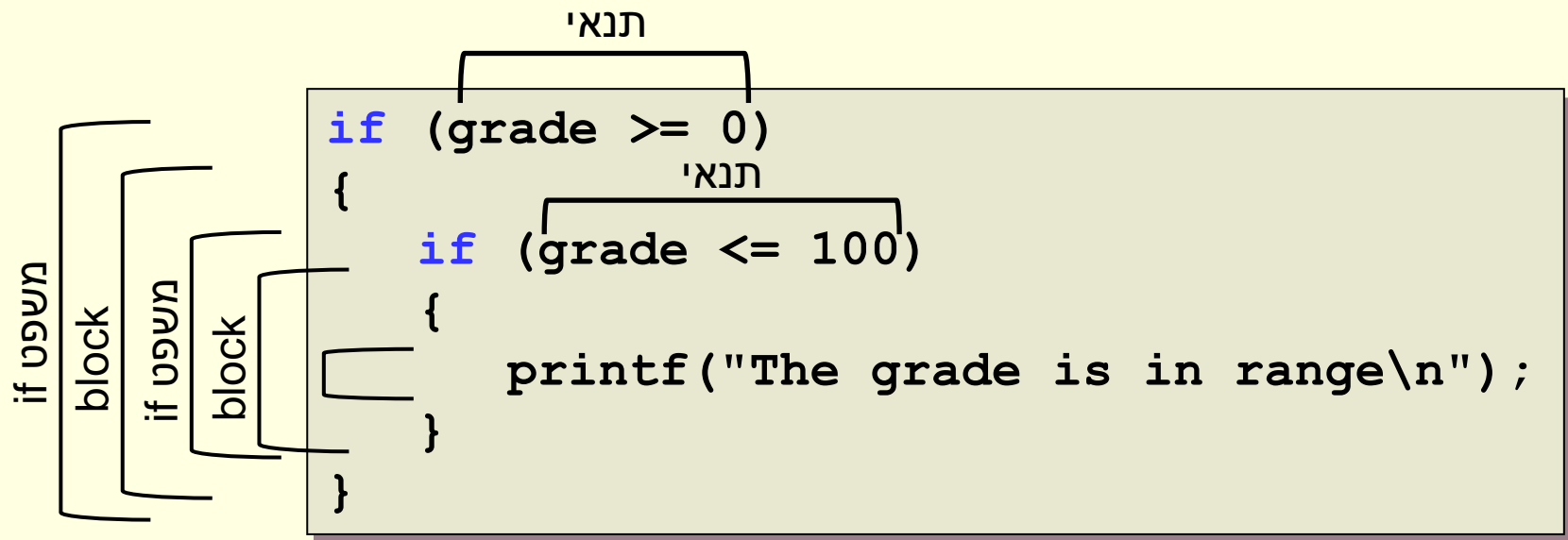
הדפס את המספר הגדול  
יותר למסך

# תנאים כמספרים

- התנאי הוא ביטוי (בעל ערך מספרי שלם)
  - לא נכון (false)  $\Leftrightarrow 0$
  - נכון (true)  $\Leftrightarrow$  מספר שונה מ-0
- ניתן להשתמש בכל ביטוי כתנאי
- רצוי להשתמש בתנאים בוליאניים (נכון/לא נכון)
- דוגמא:
  - if (a) – המשפט יבוצע כתלות בערך של a
  - if (x + 3) – המשפט יבוצע רק אם ערכו של x שונה מ-3, עדיף להשתמש בביטוי if (x != -3)

# תנאים מורכבים – קינון תנאים

- לפעמים נדרשים תנאים מורכבים:
  - ציון צריך להיות גדול או שווה ל-0
  - ציון צריך להיות קטן או שווה ל-100



# תנאים מורכבים - פעולות לוגיות

- ניתן לשלב תנאים בעזרת הפעולות הלוגיות:
  - ! (שלילה  $\Leftrightarrow$  התנאי צריך לא להתקיים)
  - && (גם  $\Leftrightarrow$  שני התנאים צריכים להתקיים)
  - || (או  $\Leftrightarrow$  מספיק שאחד התנאים יתקיים)

| <code>a&amp;&amp; b</code> | true  | false |
|----------------------------|-------|-------|
| true                       | true  | false |
| false                      | false | false |

| <code>a    b</code> | true | false |
|---------------------|------|-------|
| true                | true | true  |
| false               | true | false |

- סדר הקדימות הוא:
  1. NOT
  2. AND
  3. OR

# תנאים מורכבים – פעולות לוגיות

- לפעמים נדרשים תנאים מורכבים:
  - ציון צריך להיות גדול או שווה ל-0
  - ציון צריך להיות קטן או שווה ל-100

```
if (grade >= 0 && grade <= 100)
{
 printf("The grade is in range\n");
}
```

תנאי

תנאי

משפט if

block

# פעולות לוגיות - דוגמה

```
#include <stdio.h>
int main()
{
 double exam, exercises, final;
 printf("Enter exam grade and exercises grade");
 scanf("%lf %lf", &exam, &exercises);

 if ((exam >= 60) && (exercises >= 60))
 {
 final = 0.8 * exam + 0.2 * exercises;
 printf("You passed - your grade is %g \n", final);
 }
 else
 {
 printf("You failed!\n");
 }

 return 0;
}
```

# משפט if - הרחבה

- ראינו ביצוע מותנה של משפט אם מתקיים תנאי
- מה קורה אם בהתקיים התנאי נרצה לבצע משפט מסוים ואם אינו מתקיים משפט אחר?

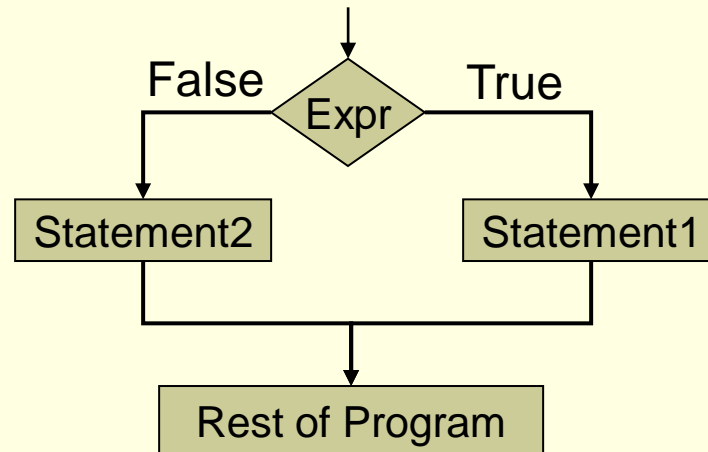
```
if (grade >= 0 && grade <= 100)
{
 printf("The grade is in range\n");
}
if (!(grade >= 0 && grade <= 100))
{
 printf("The grade is not in range\n");
}
```



# משפט if-else

```
if (expression)
 statement1
else
 statement2
```

- אם התנאי מתקיים יתבצע משפט 1 אחרת יתבצע משפט 2



# משפט if-else

```
if (grade >= 0 && grade <= 100)
{
 printf("The grade is in range\n");
}
else
{
 printf("The grade is not in range\n");
}
```

# דוגמה – בדיקה האם מספר הוא זוגי

```
#include <stdio.h>

int main()
{
 int num = 0;

 printf("Enter a number\n");
 scanf("%d", &num);

 if (num %2 == 0)
 printf("This number is even\n");
 else
 printf("This number is odd\n");

 return 0;
}
```

- משתנים
  - משפטי הגדרה והשמה
  - ביטויים
  - יצירת ביטויים מורכבים בעזרת אופרטורים
- בקרת זרימה
  - משפט if (ו-if-else)
  - תנאים כביטויים
  - אופרטורים לוגיים

# עוד פעולת תנאי - : ?

- משמשת לרישום מקוצר של תנאי באופן הבא:

; ביטוי2 : ביטוי1 ? תנאי

- אם התנאי מתקיים אז נבחר הביטוי הראשון. אחרת, נבחר הביטוי השני.
- דוגמה:

$\text{max} = (a > b) ? a : b ;$

# מה אם יש יותר משני מצבים?

• אפשר לרשום:

```
if (condition)
 command;
else
 if (condition)
 command;
 else
 if (condition)
 command;
 else
 command;
```

(אפשר להחליף פקודה בבלוק של פקודות)

# בחירה בין יותר משני מצבים - דוגמא

```
if (grade >= 90)
 printf("A\n");
```

האם הציון מעל 90?

```
else
```

```
if (grade >= 80)
 printf("B\n");
```

האם הציון מעל 80?

```
else
```

```
if (grade >= 70)
 printf("C\n");
```

האם הציון מעל 70?

```
else
```

```
 printf("Failed\n");
```

# בחירה בין יותר משני מצבים - switch

- פקודת ה-switch משמשת להשוואה עם ערכים קבועים מראש
- במקרים כאלה נוחה יותר לשימוש מ-if-else מרובים



# דוגמה switch

```
char grade;
scanf("%c", &grade);

switch (grade)
{
case 'A':
 printf("90 - 100\n");
 break;
case 'B':
 printf("80 - 89\n");
 break;
case 'C':
 printf("70 - 79\n");
 break;
default:
 printf("Failed\n");
}
```

- הערך עליו מופעל ה- switch חייב להיות מספר שלם
- הפקודה בודקת את הערך מול כל אחד מה- cases לפי סדר
- כל הפקודות החל מה- case המתאים ועד סוף ה- switch מבוצעות בזו אחר זו
- אם מופיעה הפקודה **break** קופצים לפקודה הראשונה שאחרי ה- switch
- אם לא נמצא case מתאים יבוצע ה- **default** (אם קיים)

# מבנה כללי של switch

```
switch (tav)
{
default:
 printf("Not a digit\n");
 break;
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
 printf("A digit\n");
}
```

- אם לא שמים break בין מספר cases אז אותן פקודות יבוצעו עבור מקרים שונים
- default לא חייבת להיות המקרה האחרון
- אסור שיופיעו שני cases בעלי אותו הערך