קורס יסודות התכנות בשפת C

פרק 14

סיביות/ביטים ובתים Bits & Bytes

ד"ר שייקה בילו

יועץ ומרצה בכיר למדעי המחשב וטכנולוגית מידע

מומחה למערכות מידע חינוכיות, אקדמיות ומנהליות

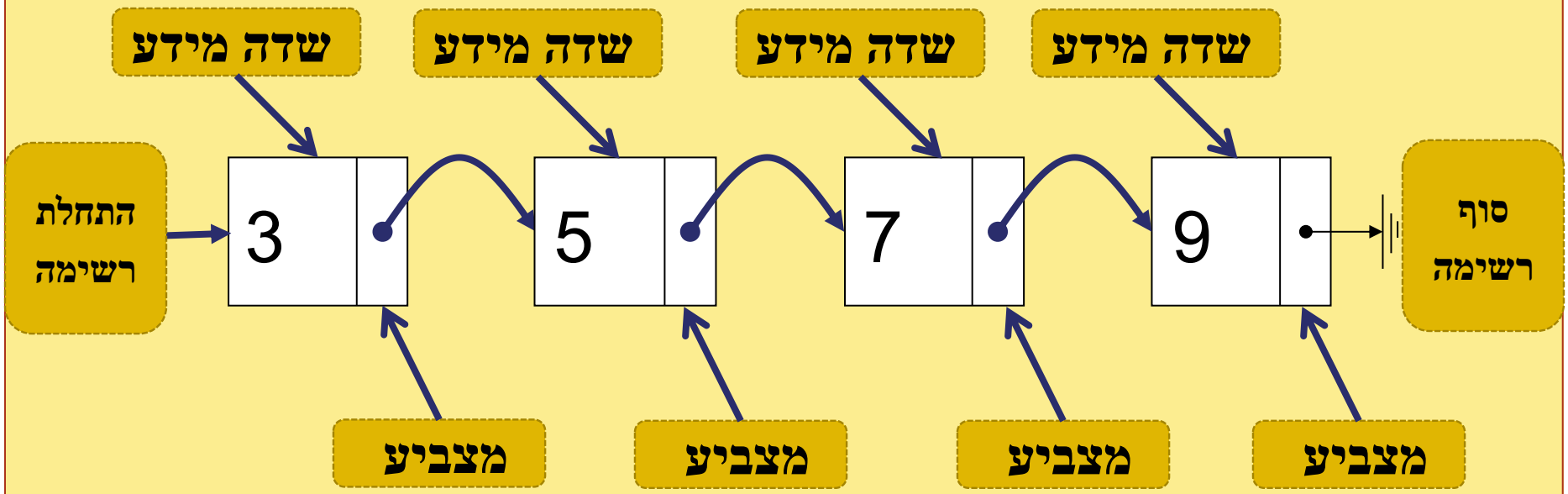
חזרה - רשימות מקושרות

- ידוע לכל מתכנת כי במערכות רבות קיימת תחלופה תכופה של אובייקטים (מחיקה והוספה בקצב מהיר).
  - למשל במערכת לניהול ספרים בספריה, מערכת לרישום סטודנטים באוניברסיטה, מערכת מעקב במצלמות כביש 6, מערכות ניווט וכו'.
- במידה ומייצגים את הנתונים באמצעות מערך, יש להעתיק את כל המערך מחדש כל פעם כשצריך להוסיף משהו, ו-"לצופף" את כל המערך כשצריך למחוק משהו מהאמצע.
  - זה לא יעיל.
  - זה לא נוח.
  - זה לא סביר.

חזרה - רשימות מקושרות

- לעבודה עם רשימות גדולות שיש בהן שינויים תכופים משתמשים ברשימה מקושרת במקום להשתמש במערך.
- בניגוד למערך, שהוא אוסף של תאים רצופים, רשימה מקושרת היא אוסף של תאים לא רצופים, שכל אחד מהם כולל גם מצביע לתא שאחריו ברשימה.
  - כפי שאמרנו, זה מקל מאוד על ביצוע תוספות ומחיקות, בלי צורך לשנות את שאר תאי הרשימה.
- כדי לעבור על הרשימה מספיק להחזיק מצביע לתא הראשון שלה, וממנו להתקדם לתא הבא, וכן הלאה.
- המוסכמה היא שהתא האחרון ברשימה מצביע על NULL.

חזרה - רשימה מקושרת של מספרים



לעומת מערך:

3	5	7	9

חזרה - הוספת איברים לרשימה

- מעצם טבעה של רשימה מקושרת (שיש בה תוספות ומחיקות), מספר האיברים בה לא ידוע מראש ואין עליו חסם.
- אנחנו נדרשים להוסיף איברים במהלך ריצת התוכנית.
- תוספת איבר לרשימה מקושרת תיעשה בעזרת הקצאה דינאמית שלו. הקצאה דינאמית של איבר אחד נעשית במקרה זה ע"י:

```
Item *ptr;
ptr = (Item *) malloc (sizeof(Item));
```

- נתאר פונקציה להוספת איבר כזה לרשימה.

חזרה - מימוש רשימות מקושרות ב-C

- בניגוד למערך, רשימה מקושרת איננה טיפוס משתנה שקיים בשפת C.
- כדי לשמור נתונים ברשימה מקושרת, עלינו לבנות אותה בעצמנו, ולממש בעצמנו את הפעולות אותן נבקש לבצע על איבריה (כמו הוספת איבר, מחיקת איבר, חיפוש איבר, וכו').
- לעינו להקפיד כי האיבר האחרון ברשימה מצביע ל- NULL ואילו האיבר הראשון מצביע לשני אחרת אין לנו בעצם רימה מקושרת.

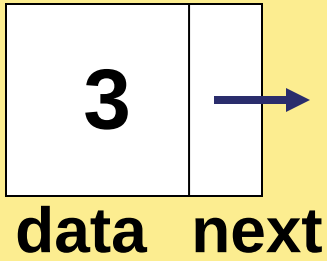
חזרה - מימוש רשימות מקושרות ב-C

- רשימה מקושרת היא למעשה פשוט אוסף האיברים שלה (שמחוברים ביניהם).
- צריך להראות איך לממש איבר אחד ברשימה ואיך לקשר את האיברים.
- כפי שאמרנו קודם, כדי לעבוד עם רשימה מקושרת יספיק לשמור את כתובת האיבר הראשון שלה, כי ממנו אפשר להתקדם לאחרים.
- קיימות רשימות מקושרות דו כיווניות וסיבוביות בהן האיבר האחרון מצביע על הראשון וכך נוצר מעגל איברים.

חזרה - מימוש רשימות מקושרות ב-C

- איבר ברשימה ייוצג על-ידי מבנה שיכיל לפחות שני שדות:
  - שדה אחד יכיל מידע.
  - שני יכיל מצביע לאיבר הבא.
- אז עבור הדוגמא של רשימת מספרים שלמים נגדיר איבר ע"י:

```
typedef struct Item
{
    int data;
    struct Item *next;
}Item;
```



  - השדה data יכיל את המספר שהאיבר מכיל.
  - השדה next יכיל מצביע לאיבר הבא ברשימה. (אם אין איבר נוסף, כלומר זה האיבר האחרון, אז ערך המצביע הזה יהיה NULL)

חזרה - מציאת איבר ברשימה


```
Item *Find(Item *head, int value)
{
   Item *check = head;
   while ((check != NULL)  && (check->data !=
                 value))
         check = check->next;
   return check;
}
```

הפונקציה מקבלת מצביע לראש הרשימה וערך מבוקש

כל עוד לא הגענו לסוף ולא מצאנו את הערך המבוקש – מתקדמים ברשימה.

כשהגענו לערך המבוקש או לסוף, מחזירים מצביע למקום שהגענו אליו

(באופן דומה יכולנו לחפש את האיבר הראשון שמקיים תנאי כלשהו)

חזרה - שחרור רשימה מקושרת

- בסיום השימוש ברשימה נרצה לשחרר את כל הזיכרון שהיא משתמשת בו.
  - בשונה ממערך שהוקצה דינאמית, free של ראש הרשימה משחרר רק את האיבר הראשון.
  - כל אחד מאיברי הרשימה הוקצה ע"י malloc נפרד, ו- free משחרר את מה שהוקצה בפקודת malloc אחת בדיוק.
- נכתוב פונקציה שמקבלת מצביע לראש הרשימה ועוברת ומשחררת את כל האיברים שהוקצו על ידי פונקציית ההוספה.

שאלות על השעור הקודם?

Bits and Bytes - סיביות ובתים

- נושאים
  - מבוא לסיביות ובתים
  - מדוע להשתמש בסיביות וביטים? מדוע לא בסיס 10?
  - ייצוג מידע בינארי / הקסה-דצימלי
    - ייצוג של בתים
      - מספרים
      - תווים ומחרוזות
      - פקודות (instructions)
  - פעולות ברמת הביטים
    - לוגיקה בוליאנית
    - כיצד הדבר נראה בתוכניות C ?

מבוא לסיביות ובתים – Bits and Bytes

- זיכרון של מחשבים, ושל מיקרו-בקרים, הוא למעשה רצף אחד ארוך של ביטים: "תאים" זעירים שכל אחד מהם יכול להיות באחד משני מצבים מוגדרים מראש (מצבים אותם אנחנו מכנים "0" ו-"1"). כל משתנה, כל ערך שאנחנו מציינים בתוכנה שלנו הוא בסופו של דבר קטע קצר על פני הרצף הזה.

מבוא לסיביות ובתים – Bits and Bytes

- רוב שפות התכנות המודרניות עושות מאמצים להסתיר את העובדה הזאת, אבל היא נדרשת כשעובדים עם מיקרו-בקרים.
- נושא הביטים מתחלק לשני תת-נושאים עיקריים: ייצוג בינארי, ולוגיקה בוליאנית והם נושאים טכניים. אף על פי כן, מבין כל הנושאים שניים אלה הנושאים הכי קריטיים.

מבוא לסיביות ובתים – Bits and Bytes

- ביטים כמייצגי מספרים:

ביט הוא תא שיכול להכיל 0 או 1. אם ניקח רצף קצר של ביטים ממקום אקראי בזיכרון, נראה משהו דומה לזה: 0111001010011010. מה זה אומר?

איך דבר כזה מתורגם למספרים שאנחנו מכירים?

מבוא לסיביות ובתים – Bits and Bytes

- כשאנחנו כותבים מספר בבסיס עשר היומיומי והמוכר, הספרה הימנית ביותר היא "ספרת האחדות", והיא יכולה להיות בטווח 0-9.
- הספרה שמשמאלה היא באותו טווח, אך המשמעות שלה אחרת: זו כבר "ספרת העשרות", שמוכפלת בעשר. הספרה הבאה מוכפלת במאה, זאת שאחריה באלף וכך הלאה.
- נכתוב את המכפילים הללו אחד אחרי השני:

1 – 10 – 100 – 1,000 – 10,000 – 100,000 – 1,000,000 – …

מבוא לסיביות ובתים – Bits and Bytes

- לא צריך להיות גאון במתמטיקה כדי לזהות שכל המספרים האלה הם חזקות של 10:

$$\ldots - 10^6 - 10^5 - 10^4 - 10^3 - 10^2 - 10^1 - 10^0$$

- כלומר, כל ספרה מוכפלת למעשה בעשר-בחזקת-המיקום-שלה, כאשר ספירת המיקומים מתחילה מאפס. לדוגמה, אם המספר הוא 321, הספרה 1 שנמצאת במיקום אפס תוכפל בעשר בחזקת אפס (=1).

מבוא לסיביות ובתים – Bits and Bytes

- כל זה טוב ויפה אם הטווח של כל ספרה הוא אכן 0-9, כלומר שיש עשר אופציות לכל ספרה. מה נעשה אם יגבילו אותנו לשתי אופציות בלבד, בטווח 0-1? במקום בסיס עשר, שעובד עם חזקות של עשר, אנחנו עוברים בלית ברירה לבסיס שתיים (בינארי), שעובד באותה שיטה רק עם חזקות של שתיים:

$$\ldots - 2^6 - 2^5 - 2^4 - 2^3 - 2^2 - 2^1 - 2^0$$

- נבצע את החישובים בשיטה העשרונית ונקבל

$$64 - 32 - 16 - 8 - 4 - 2 - 1$$

מבוא לסיביות ובתים – Bits and Bytes

- זאת אומרת, כדי לפענח מספר שנכתב בבסיס בינארי, נכפיל את הספרה הימנית ביותר ב-1, את זו שמשמאלה ב-2, את הבאה בתור ב-4 וכן הלאה, ונחבר הכל ביחד. לדוגמה, כמה יוצא המספר הבינארי הבא?

01111011

אם קיבלתם 123, כל הכבוד. אם לא, להלן התרגיל:

123=(1*1)+(1*2)+(0*4)+(1*8)+(1*16)+(1*32)+(1*64)+(0*128)

מבוא לסיביות ובתים – Bits and Bytes

- תרגיל כיתה קצר להפיכת מספר שלם למספר בינארי:
- קלוט את המספר הבינארי 0111011 כמחרוזת וייצר ממנו מספר שלם, את המשימה יש לבצע ע"י שימוש בפונקציה הקולטת את המספר הבינארי כמחרוזת ומייצרת ממנו מספר שלם.

אם קיבלתם 123, כל הכבוד.

אם לא, להלן התרגיל:

123=(1*1)+(1*2)+(0*4)+(1*8)+(1*16)+(1*32)+(1*64)+(0*128)

מבוא לסיביות ובתים – Bits and Bytes

- בכל שפות התכנות, יחידת הנתונים הבסיסית ביותר היא הבייט ((byte) שכולל שמונה ביטים.
- מה טווח המספרים שאפשר לייצג באמצעות בייט אחד? במילים אחרות, כמה זה 00000000 וכמה זה 11111111?
- אחסוך לכם את החישוב הפעם: התשובות הן 0 ו-255.

מבוא לסיביות ובתים – Bits and Bytes

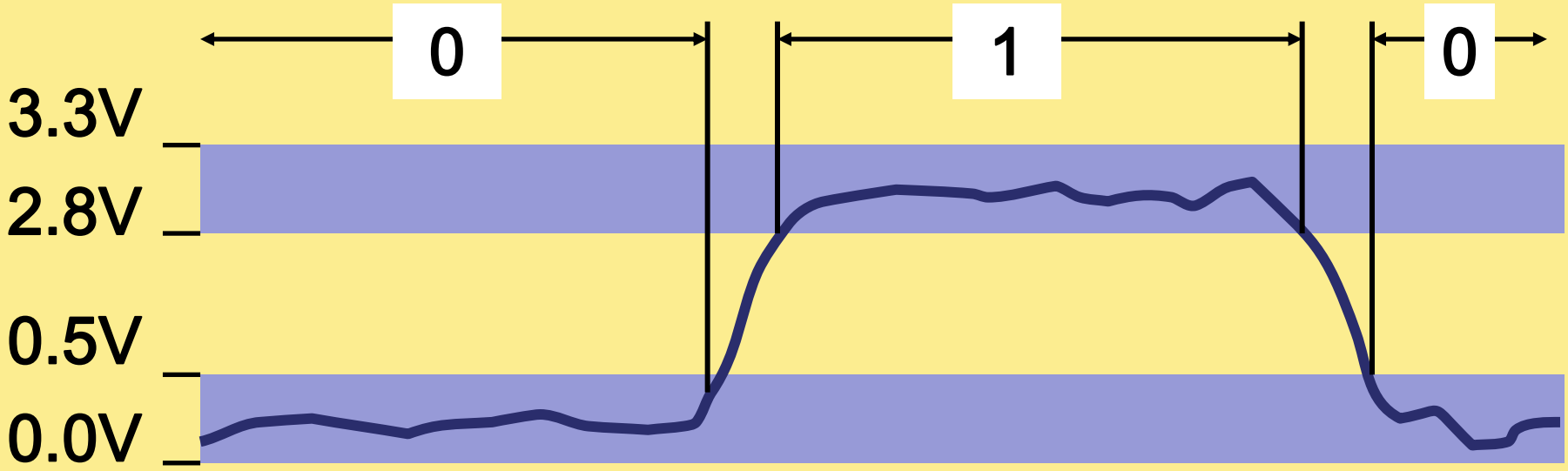
- נניח שיש לנו משתנה מטיפוס בייט שערכו 255, ואנחנו מוסיפים לו אחד, מתמטית, זה כמו להוסיף 1 ל-999. כל הספרות מתאפסות, ונוסף לנו 1 משמאל.
- אבל מכיוון שהבייט מגביל אותנו ל-8 ביטים ואין אפשרות להוסיף 1 משמאל, אנחנו נשארים עם ביטים שכולם אפסים בלבד.
- חזרנו לאפס! אם נוסיף 2 ל-255 נקבל 1, וכן הלאה.
- באותו אופן, אם ננסה להפחית 1 מבייט שערכו אפס (יש לזכור כי בייט אינו יכול לקבל ערכים שליליים) נחזור ל-255.

מדוע מחשבים לא עובדים בבסיס 10 ?

- המחשב האלקטרוני הראשון ENIAC השתמש ב- 10 שפורפרות ואקום לכל ספרה, סרבול אדיר ובזבוז זמן וחשמל.
- קושי טכני פיזי חשמלי הנובע מהקושי לאגור ולהעביר מידע בכמה רמות מתח שונות במקביל:
  - דרוש דיוק גבוה כדי לקודד 10 רמות שונות של אינפורמציה על מוליך אלקטרוני אחד.
- אורך מורכב ומסובך מאד לבצע חישובים לוגיים:
  - חיבור, חיסור, כפל, חילוק, העלאה בחזקה והוצאת שורש ו...

ייצוג בינארי חשמלי

- מימוש בחומרה
  - קל לאגור עם אלמנטים bi-stable
  - העברה אמינה גם כשהמוליכים לא אמינים ומדויקים, בגלל מתח נמוך



ייצוג בינארי

- ייצוג בינארי (בסיס 2)
  - נייצג את 15213_{10} עם 11101101101101_2
- בהמשך נלמד איך מייצגים שברים:
  - נייצג את $1.5213 \text{ X } 10^4$ עם $1.\underbrace{1101101101101}_{\text{13 סיביות}}{}_2 \text{ X } 2^{13}$
  - נייצג את 1.20_{10} עם $1.0011001100110011[0011]\ldots_2$

ייצוג מספרים בבסיסים שונים

- ייצוג hexadecimal:
  - בסיס 16
  - ספרות 0- 9 ואותיות 'F' - 'A'
- ב – C כתוב $FA1D37B_{16}$ כך:
  - 0xFA1D37B או
  - 0xfa1d37b

ייצוג מספרים בבסיסים שונים

דצימאלי	ייצוג בינארי	הקס-דצימאלי
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

ייצוג מספרים בבסיסים שונים

דצימאלי	ייצוג בינארי	הקס-דצימאלי
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

דוגמה: טווח של ערכי בתים

- בית = 8 סיביות
- אפשרויות תצוגת מספרים לפי בסיסים:

1. Binary: from 00000000_2 to 11111111_2
2. Decimal: from 0_{10} to 255_{10}
3. Hexadecimal: from 00_{16} to FF_{16}

שאלות?

מילים (machine words)

- לכל מחשב יש 'רוחב מילה'
  - הגודל המוקצה למידע מסוג integer (כולל כתובות).
- רוב המחשבים המתקדמים היום כבר עובדים עם פס כתובות ברוחב 64 סיביות (8 בתים)
  - מרחב הכתובות $1.8 \text{ X } 10^{19} \approx 18$ הקסה בתים.
  - מרחב הכתובת נמדד בהקסה ולא בג'יגה כמו ברוחב פס של 32 סיביות שהגביל את המרחב ל- 4.3GB
  - רוחב פס זה מספיק בהחלט, כרגע, לתנועה מרובת מידע במערכות הן של מחשבים מרובי ליבות והן אחרים.

ארגון הזיכרון לפי בתים

- תוכניות מתייחסות לכתובות 'מדומות' (וירטואליות)
- באופן אבסטרקטי:

1. מערך גדול של בתים
2. בפועל ממומש עם היררכיה שלמה של סוגי זיכרונות
3. מהדר (compiler) + רכיבי חומרה קובעים את המיפוי לזיכרון הפיזי.
4. נראה את ייצוג הכתובות במערך זה.

ייצוג מידע

- גודל בבתים של אובייקטים שונים של C:

C *Data Type*	Compaq Alpha	Typical 32bit/64bit	Intel IA32
int	4	4	4
long int	8	4	4
char	1	1	1
short	2	2	2
float	4	4	4
double	8	8	8
long double	8	8	10/12
char *	8	4	4

סידור המידע

- ניתן לסדר את הבתים בתוך מלים מימין לשמאל או להפך...
- מה מקובל ? אין הסכמה
- Sun's, Mac's are "Big Endian" machines
- הבית הימני (least significant byte) בכתובת הגבוהה
- Alphas, PC's are "Little Endian" machines
- הבית הימני בכתובת הנמוכה

דוגמה

- Big Endian
  - הבית הימני בכתובת הגבוהה !!!
- Little Endian
  - הבית הימני בכתובת הנמוכה !!!
  - `x = 0x01234567`
  - למשתנה x יש ייצוג ב 4 בתים.
  - נניח שהכתובת של x (כלומר &x) היא 0x100

Big Endian

		0x100	0x101	0x102	0x103		
		01	23	45	67		

Little Endian

		0x100	0x101	0x102	0x103		
		67	45	23	01		

כיצד נראה איך המידע מיוצג

- תוכנית להדפסת ייצוג בבתים:

```
#include <stdio.h>
#define SIZE 10
void show_bytes(int start[])
{
  int i;
  for (i = 0; i < SIZE; i++)
      printf("0x%p\t0x%.2x\n", start+i, start[i]);
  printf("\n");
}
```

Printf directives:

%p: Print pointer

%x: Print Hexadecimal

כיצד נראה איך המידע מיוצג

- תוכנית להדפסת ייצוג בבתים:

```
int main()
}
        int start[SIZE]={0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f};
        show_bytes(start);
return 0;
{
```

ייצוג פקודות מכונה

- פקודות מקודדות כבתים
  - מחשבי RISC כגון Sun משתמשים באוסף קטן של פקודות:
    - 4 בתים מספיקים לייצוג כל הפקודות.
- מחשבי CISC כגון ה - PC משתמשים בקבוצה גדולה של הוראות:
    - נעזרים בגודל משתנה של פקודה.

ייצוג פקודות


```
int sum(int x, int y)
{
  return x+y;
}
```

לכל פקודה יש קוד מספרי

Sun sum	PC sum
81	55
C3	89
E0	E5
08	8B
90	45
02	0C
00	03
09	45
	08
	89
	EC
	5D
	C3

- דוגמה זאת מתורגמת למספר פקודות בסיסיות בשפת מכונה.
- בדוגמה זאת מחשבי Alpha, Sun משתמשים ב 4 בתים.
- בדוגמה זאת מחשבי PC משתמשים בפקודות באורכים של 1,2,3 בתים.

מחשבים שונים משתמשים בפקודות שונות וקידוד שונה של הפקודות.

אין סיכוי לתאימות!

קידוד הפקודה sum

שאלות?

מה עכשיו ?

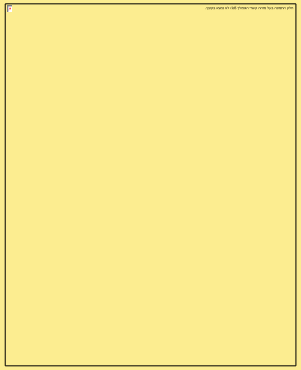
- עכשיו, כשראינו שאת כל סוגי המידע שהמחשב משתמש בהם, ניתן לייצג על ידי שימוש בתווים 0 ו 1, ...
- נרצה לראות איך המחשב יכול לבצע חישובים עם מידע זה.
- לשם כך, עלינו להיזכר בלוגיקה פסוקית.
- לוגיקה פסוקית == לוגיקה בוליאנית מבוססת על מצבי אמת שקר ובדיקתם ביחס שבין שני איברים ומעלה ע"י התנאים הבוליאניים הבאים:

- And - וגם
- Or - או
- Xor - או בלעדי
- Not - שלילה

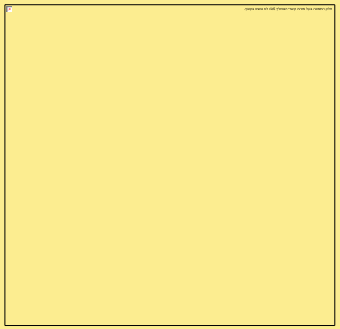
לוגיקה פסוקית (בוליאנית)

- פותחה על ידי George Bool במאה ה - 19
  - נקודד 'אמת' כ – 1 ו 'שקר' כ - 0

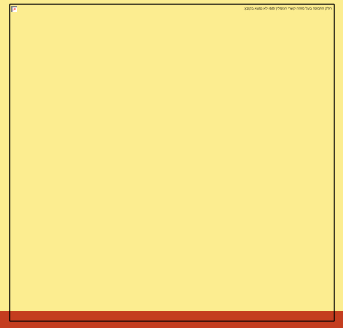
And



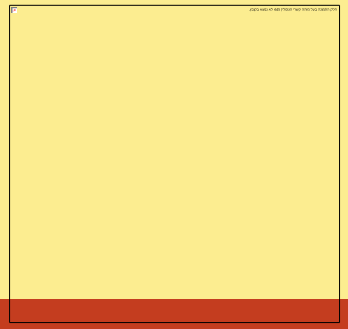
Or



Not



Exclusive-Or (Xor)



נשווה אלגברה בוליאנית לאלגברה אריתמטית

- נציב '|' = '+' ו- '&' = '*'.
- תרגיל: עבור כל אחד מהביטויים הבאים – קבעו אם הביטוי המקביל נכון.
- קומוטטיביות(תכונה של פעולות בינאריות) :

$$A \mid B = B \mid A \qquad A + B = B + A$$

$$A \,\&\, B = B \,\&\, A \qquad A * B = B * A$$

- אסוציאטיביות (תכונה של פעולה בינארית המיישמת את חוק הקיבוץ):

$$(A + B) + C = A + (B + C)$$

$$(A * B) * C = A * (B * C)$$

פעולות על סיביות

- שפת C מאפשרת לבצע פעולות על סיביות (bitwise operations), שהן פעולות על הסיביות של האופרנדים:
  - פעולת AND המסומנת על-ידי האופרטור &.
  - פעולת OR המסומנת על-ידי האופרטור |.
  - פעולת NOT המסומנת על-ידי האופרטור ~.
  - פעולת XOR המסומנת על-ידי האופרטור ^ .
  - פעולת הזזה שמאלה (הגדלה) המסומנת על-ידי האופרטור << .
  - פעולת הזזה ימינה (הקטנה) המסומנת על-ידי האופרטור >> .
- כל האופרטורים הפועלים על סיביות הם אופרטורים בינאריים (המצפים לקבל שני אופרנדים), למעט האופרטור NOT שהוא אופרטור(המצפה לקבל אופרנד אחד).

הפעולה AND

- הפעולה AND על סיביות מסומנת על-ידי &, והיא דומה לפעולה AND הבוליאנית המסומנת על-ידי &&.
- טבלת האמת של הפעולה &:

```
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

- כלומר, פעולת & בין שתי סיביות תחזיר 0 אם לפחות אחת מהסיביות היא 0; ותחזיר 1 רק אם שתי הסיביות הן 1.

הפעולה AND

- אם נבחן את טבלת האמת של הפעולה AND, נבחין בתכונות הבאות:
  - כשמפעילים על סיבית מסוימת b את הפעולה & עם סיבית 1, אז הסיבית לא משתנה.

$$b \,\&\, 1 = b$$

  - כשמפעילים על סיבית מסוימת b את הפעולה & עם סיבית 0, אז הסיבית מתאפסת.

$$b \,\&\, 0 = 0$$

הפעולה AND

- המסקנה היא שניתן להשתמש בפעולה & על מנת למַסֵך (to mask) סיבית מסוימת, ולבודד אותה משאר הייצוג של המספר.
- לדוגמא: נניח שבמשתנה num מאוחסן מס' שלם, ואנחנו מעוניינים לדעת מהי הסיבית הרביעית מימין. הנה קטע קוד מתאים:

```
if (num & 8 != 0)
    printf ("הסיבית הרביעית מימין היא דלוקה");
else
    printf ("הסיבית הרביעית מימין היא כבויה");
```

הפעולה OR

- הפעולה OR על סיביות מסומנת על-ידי |, והיא דומה לפעולה OR הבוליאנית המסומנת על-ידי ||.
- טבלת האמת של הפעולה |:

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

- כלומר, פעולת | בין שתי סיביות תחזיר 1 אם לפחות אחת מהסיביות היא 1; ותחזיר 0 רק אם שתי הסיביות הן 0.
- מה יהיה הפלט של קטע הקוד הבא:

```
int num = 72 | 184;
printf ("%d", num);
```

פלט יהיה: 248.

הפעולה OR

- אם נבחן את טבלת האמת של הפעולה OR, נבחין בתכונות הבאות:
  - כשמפעילים על סיבית מסוימת b את הפעולה | עם סיבית 1, אז הסיבית נדלקת.

$$b \mid 1 = 1$$

  - כשמפעילים על סיבית מסוימת b את הפעולה | עם סיבית 0, אז הסיבית נותרת ללא שינוי.

$$b \mid 0 = b$$

הפעולה OR

- המסקנה היא שניתן להשתמש בפעולה | על מנת להדליק סיביות מבוקשות, מבלי לשנות סיביות אחרות.
- לדוגמא: נניח שבמשתנה num מאוחסן מס' שלם, ואנחנו מעוניינים להדליק את הסיבית הרביעית מימין.
- הנה הוראה מתאימה:

```
num = num | 8;
```

- קיימת גם ההוראה המקוצרת =&.

הפעולה OR

- תכונות נוספות של הפעולה OR:
  - כשמפעילים על סיבית מסוימת b את הפעולה | עם עצמה, אז הסיבית לא משתנה.

$$b \mid b = b$$

  - כשמפעילים על סיבית מסוימת b את הפעולה | עם הסיבית ההפוכה, אז הסיבית נדלקת.

$$b \mid \sim b = 1$$

הפעולה OR

- מה יהיה הפלט של קטע הקוד הבא:

```
int num1=11;
int num2=num1 | num1, num3=num2 &
  num2;
printf ("%d %x %o\n", num2, num2, num2);
printf ("%d %x %o\n", num3,num3,num3);
```

הפעולה NOT

- הפעולה NOT על סיביות מסומנת על-ידי ~, והיא זהה לפעולה NOT הבוליאנית המסומנת על-ידי !.
- טבלת האמת של הפעולה ~:

~0 = 1

~1 = 0

- כלומר, כשמבצעים פעולת ~ על סיבית, היא מתהפכת, כלומר, ערכה הופך מאפס לאחד, או מאחד לאפס.

הפעולה NOT

- מה יהיה הפלט של קטע הקוד הבא:

```
int num = 72;
printf ("%d", ~num);
```

הפלט במקרה זה יהיה 73-.

- ומה יהיה הפלט אם נרשום %u במקום %d?
- הפלט יהיה 65,463 (בהנחה שגודלו של int הוא 2 בתים ולא 4 בתים).

הפעולה NOT

- ברור שמתקיים ~~b == b .
- אם a ו-b הן סיביות, אז מתקיימים גם חוקי דה-מורגן:

~(a & b) == ~a | ~b

~(a | b) == ~a & ~b

- מה יהיה הפלט של קטע הקוד הבא?

```
char num = 13;
printf ("%d %d", num &~num,num| ~num);
```

הפעולה XOR

- הפעולה XOR (exclusive or) מסומנת על-ידי ^. טבלת האמת של הפעולה XOR ('או' מוציא) זהה לטבלת האמת של הפעולה OR ('או' מכיל), למעט במקרה ששני האופרנדים הם 1. במקרה כזה – XOR תחזיר 0.
- טבלת האמת של הפעולה ^:

```
0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0
```

- כלומר, פעולת ^ בין שתי סיביות תחזיר 1 אם הסיביות הן שונות זו מזו, ותחזיר 0 אם הן זהות.

הפעולה XOR

- מה יהיה הפלט של קטע הקוד הבא:

```
unsigned int num = 72 ^ 184;
printf ("%u", num);
```

- כלומר הפלט יהיה 240.

הפעולה XOR

- אם נבחן את טבלת האמת של הפעולה XOR, נבחין בתכונות הבאות:
  - כשמפעילים על סיבית מסוימת b את הפעולה ^ עם סיבית 1, אז הסיבית מתהפכת.

$$b \wedge 1 = \sim b$$

  - כשמפעילים על סיבית מסוימת b את הפעולה ^ עם סיבית 0, אז הסיבית נותרת ללא שינוי.

$$b \wedge 0 = b$$

הפעולה XOR

- מה יהיה הפלט של קטע הקוד הבא?

```
int num1 = 72;
int num2 = num1 ^ -1;
printf ("%d %u", num2, num2);
```

- בדומה לפעולות =& , =| קיימת גם הפקודה =^ .

הפעולה XOR

- תכונות נוספות של הפעולה XOR:
  - כשמפעילים על סיבית מסוימת b את הפעולה ^ עם עצמה, אז הסיבית מתאפסת.

$$b \wedge b = 0$$

  - כשמפעילים על סיבית מסוימת b את הפעולה ^ עם הסיבית ההפוכה, אז הסיבית נדלקת.

$$b \wedge \sim b = 1$$

הפעולה XOR

- מה יהיה הפלט של קטע הקוד הבא:

```
int num = 11;
int num2=num1^num1,num3=num1^~num1;
printf ("%d %d", num2, num3);
```

כתבו הוראה אחת בלבד ההופכת (~) את הסיבית שלישית מימין של המשתנה num:

```
num = num ^ 4;
```

הפעולה XOR

- שימוש מעניין בפעולה XOR הוא על מנת לממש פונקציית החלפה (swap), אך ללא שימוש במשתנה עזר:

```
void swap (int *num1, int *num2)
{
        *num1 = *num1 ^ *num2;
        *num2 = *num1 ^ *num2;
        *num1 = *num1 ^ *num2;
}
```

הפעולה XOR

- או על-ידי שימוש בהוראות המקוצרות:

```
void swap (int *num1, int *num2)
{
        *num1 ^= *num2;
        *num2 ^= *num1;
        *num1 ^= *num2;
}
```

הזזה

- בשפת C קיימים אופרטורים המאפשרים להזיז את סיביות הביטוי ימינה ושמאלה.
- זהו האופרטור >> (המבצע הזזה בסיביות שמאלה) והאופרטור << (המבצע הזזה בסיביות ימינה). למקומות שיתפנו ייכנסו אפסים.
- שני האופרטורים מקבלים שני אופרנדים:
  - האופרנד השמאלי זהו הביטוי שאת הסיביות שלו יש להזיז.
  - האופרנד הימני זהו מס' ההזזות שצריך לבצע.

הדגמה

- מה יהיה הפלט של קטע הקוד הבא?

```
printf ("%d", 7 << 2);
```

- מה מבצע קטע הקוד הבא?

```
int count = 0;
scanf ("%d", &a);
while (a != 0)
{
    count += a & 1;
    a=a>>1;  /* אפשר היה לרשום בקיצור: a >>= 1 */
}
printf ("%d", count);
```

הזזה

- מה לדעתכם יהיה הפלט של קטע התכנית הבא?
- האם זה יהיה הפלט בכל סביבת עבודה?
- ניתן להשתמש באופרטורים של הזזה על מנת לממש ביעילות כפל בחזקות של שתיים או חלוקה בחזקות של שתיים:
  - a = a << k יכפיל את a פי 2^k.
  - a = a >> k יחלק את a פי 2^k.

```
#include <stdio.h>
int main()
{
    printf ("%x", -1 << 4);
    return 0;
}
```

נשווה אלגברה בוליאנית לאלגברה אריתמטית

- המשך ...

1. $A * (B + C) = A * B + A * C$
2. $A + 0 = A$
3. $A | A = A$
4. $A \& 1 = A$
5. $A * 0 = 0$
6. $A | (B \& C) = (A | B) \& (A | C)$
7. $-(-A) = A$
8. $A + A \neq A$

נשווה אלגברה בוליאנית לאלגברה אריתמטית

- המשך ...

1. $A \& A = A$
2. $A | (A \& B) = A$
3. $A \& (A | B) = A$
4. $A | !A = 1$
5. $A | !A \neq 0$

עוד מספר התמרות בין אופרטורים

- חוקי DeMorgan

A & B = !(!A | !B)

A | B = !(!A & !B)

- xor על ידי | ו &:

A ^ B = (!A & B) | (A & !B)

A ^ B = (A | B) & !(A & B)

מספר זהויות: & ו - ^

- תרגיל: מצא את הזהויות השגויות

1. A ^ B = B ^ A
2. A & B = B & A
3. (A ^ B) ^ C = A ^ (B ^ C)
4. (A & B) & C = A & (B & C)
5. A & (B ^ C) = (A ^ B) & (A ^ C)
6. A ^ 0 = 0
7. A & 1 = A
8. A & 0 = 0
9. A ^ A = A

חיבור וקטורים של ביטים


```
 01001101 +
 10000100
 ----------
 11010001

  0010101 +
  1101100
 ----------
  0000001
```

מביטים לוקטורים של ביטים

- פעולות על וקטורים של ביטים
  - פעולות מבוצעות לחוד עבור כל ביט (bitwise)

and &

```
  01101001
& 01010101
  01000001
```

xor ^

```
  01101001
^ 01010101
  00111100
```

or |

```
  01101001
| 01010101
  01111101
```

not !

```
! 01010101
  10101010
```

ייצוג וחישוב על קבוצות

- ייצוג: וקטור ביטים ברוחב w יכול לייצג תתי קבוצות של $\{0,\ldots, w-1\}$

$a_j = 1$ if $j \in A$

01101001	{ 0, 3, 5, 6 }
76543210	
01010101	{ 0, 2, 4, 6 }
76543210	

- פעולות

&	Intersection	01000001	{ 0, 6 }
|	Union	01111101	{ 0, 2, 3, 4, 5, 6 }
^	Symmetric difference	00111100	{ 2, 3, 4, 5 }
!	Complement	10101010	{ 1, 3, 5, 7 }

פעולות על ביטים ב- C

- הפקודות ^ ,! ,| ,& קיימות ב - C
- רלבנטי לכל ה- integers:
  - long, int, short, char
  - מופעל על המספר בייצוגו הבינארי
  - מופעל בנפרד על כל ביט
  - דוגמאות על משתנה מסוג char:

זכרו:
$4_{16}=0100_2$
$1_{16}=0001_2$

- ~0x41 --> 0xBE
  - $\sim 01000001_2 \rightarrow 10111110_2$
- ~0x00 --> 0xFF
  - $\sim 00000000_2 \rightarrow 11111111_2$
- 0x69 & 0x55 --> 0x41
  - $01101001_2 \ \& \ 01010101_2 \rightarrow 01000001_2$
- 0x69 | 0x55 --> 0x7D
  - $01101001_2 \ | \ 01010101_2 \rightarrow 01111101_2$

פעולות "זִיזַה" (shifting)

- הזזה שמאלה: x << y
  - הזז וקטור x שמאלה ב- y מקומות
  - זרוק ביטים משמאל ומלא ב – 0 מימין
  - הגדלה של המספר פי שתיים בכל הזזה שמאלה
  - כמות ההזזות חייבת להיות מצוינת מצד ימין להזזות

פעולות "זִיזַה" (shifting)

- הזזה ימינה: x >> y
  - הזז וקטור x ימינה ב y מקומות
  - זרוק ביטים מימין
  - שני מקרים:
    - הזזה לוגית (logical shift): מלא ב- 0 משמאל
    - הזזה אריתמטית (arithmetic shift): שכפל הספרה השמאלית (msb). משתמשים בזה בייצוג 2's complement שעוד נלמד.

כיצד נגלה את הביט ה -i

- ערך הסיבית הימנית של int x:

```
int res;
res = x & 1;
```

- מקובל להשתמש במסכות.
- מציאת הסיבית ה- k + 1 מימין:

```
int mask;
mask = 1 << k;
res = (x & mask) != 0;
```

ובניגוד לכך: פעולות לוגיות ב - C

- פעולות לוגיות ב – C ! ,|| ,&&:
  - '0' מייצג שקר, '1' אמת
  - כל מספר שונה מ – 0 הוא 'אמת'
  - פעולה בוליאנית תמיד מחזירה 0 או 1
- דוגמאות על משתנה מסוג char:
  - `!0x41 --> 0x00`
  - `!0x00 --> 0x01`
  - `!!0x41 --> 0x01`
  - `0x69 && 0x55 --> 0x01`
  - `0x69 || 0x55 --> 0x01`

סיביות וביטים – סיכום ביניים

- הכל זה סיביות ובתים...
  - מספרים, תוכניות, טקסט
- מחשבים שונים משתמשים בייצוגים שונים
  - רוחב מילה, סדר בתים, ייצוג
- לוגיקה פסוקית מהווה כלי יסודי בהבנת פעולות המחשב
- מצד אחד, היא מאפשרת חישובים מעל וקטורים של ביטים
- מצד שני, ניתן לממש אותה בחומרה.

שאלות?

ייצוג מספרים במחשב

- כל פריט מידע במחשב נשמר כרצף של ביטים
- ביט יכול לקבל אך ורק את הערכים 0 או 1
- רצף של שמונה ביטים מהווה בית (byte) שהיא יחידת הזיכרון הקטנה ביותר בעלת כתובת
- בשפת C מוגדרים אופרטורים לעבודה עם ביטים.
- עבודה עם ביטים נעשית על משתנים המייצגים מספרים שלמים בלבד!!! (char, short, int, long)
  - ניתן להשתמש גם בגרסת המשתנה ה-unsigned של כל אחד מהם.

ייצוג מספרים בבסיסים שונים

- מספרים במחשב מיוצגים על בסיס 2 (בסיס בינארי)
- ניתן להתייחס למספר בייצוג אוקטלי (בסיס 8) ע"י הוספת 0 לפני המספר.
- לדוגמה המספר 042 באוקטלי מייצג את המספר 34 בבסיס עשרוני
- תו המרה רלבנטי הוא o%

פעולות חישוב על ביטים

- ניתן להתייחס למספר בייצוג הקסדצימלי (בסיס 16) ע"י הוספת 0x לפני המספר. לדוגמה המספר 0x42 מייצג את המספר 66 בבסיס עשרוני.
  - תו המרה רלבנטי הוא %x (או %X לקבלת אותיות גדולות)
- על מנת להשתמש באופרטורים של ביטים נדרש לייצג את המספר בייצוג בינארי
- פעולות חישוב על ביטים מבוצעות ביט מול ביט
- בדרך כלל עדיף להשתמש בגרסת **unsigned** של המשתנה

המרה מבינארי לעשרוני

- אלגוריתם המרה של מספר בבסיס בינארי למספר בבסיס עשרוני:
  - לכל ספרה בייצוג יש "משקל" שהוא 2 בחזקת האינדקס של מיקומה במספר (אינדקס מתחיל מימין ומהערך 0)

אינדקס	0	1	2	3	4	5	6	7	8
משקל	1	2	4	8	16	32	64	128	256
חישוב	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8

  - כפול כל ספרה במשקלה המתאים עפ"י מיקומה בייצוג הבינארי, הערך של המספר העשרוני הוא סכום המכפלות

המרה מבינארי לעשרוני

- לדוגמה, הייצוג הבינארי הבא:

00001101

- המיקומים הם:

128=0 ,64 =0 , 32=0 ,16=0 ,8=1 ,4=1 ,2=0 ,1=1

- הייצוג הנ"ל מייצג את המספר 13:

1*1+0*2+1*4+1*8+0*16+0*32+0*64+0*128 =

1*1 + 1*4 + 1*8 =

1+4+8 = 13

כתוב תוכנית המייצרת מספר דצימלי, בבסיס 10, ממספר בינארי הנקלט כמחרוזת.

המרה מעשרוני לבינארי

- אלגוריתם המרה של מספר בבסיס עשרוני למספר בבסיס בינארי:
  - כל עוד המספר אינו 0
  - חלק את המספר ב-2
  - שמור את השארית
- הייצוג הבינארי של המספר, משמאל לימין, הוא אוסף השאריות מהסוף להתחלה.
- כאשר יש מצב שחלק מהספרות חסרות, הן מקבלות את הערך 0.

המרה מעשרוני לבינארי

- לדוגמה – המספר 8 בייצוג בינארי הוא: 00001000

מספר	שארית
8	0
4	0
2	0
1	1
0	

ייצוג אוקטלי (בסיס 8)

- בבסיס אוקטלי קיימות רק 8 ספרות, הספרות 0-7
- כל ספרה בבסיס אוקטלי מיוצגת ע"י 3 ספרות בינאריות וזאת עפ"י הטבלה המוצגת בשקף הבא
- בהמרה של מספר מבינארי לאוקטלי – כל 3 ספרות בייצוג הבינארי הופכות לספרה אחת בייצוג האוקטלי

ייצוג אוקטלי (בסיס 8)

ספרה	ייצוג בינארי
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

- לדוגמה הייצוג הבינארי של המספר: 00001101 הוא המספר 15 בבסיס אוקטלי.
- בהמרה מאוקטלי לעשרוני ניתן להשתמש בשיטת סכום המשקלים כאשר משקלי הספרות הם (1, 8, 64 וכו')

ייצוג הקסה-דצימלי (בסיס 16)

- בבסיס הקסה-דצימלי קיימות 16 ספרות, הספרות 0-9 והספרות A-F
- כל ספרה בבסיס הקסה-דצימלי מיוצגת ע"י 4 ספרות בינאריות עפ"י הטבלה הבאה:
- בהמרה מבינארי להקסה-דצימלי – כל 4 ספרות בייצוג הבינארי הופכות לספרה אחת בייצוג ההקסה-דצימלי
- בהמרה מהקסה-דצימלי לבינארי – כל ספרה מיוצגת ע"י רביעיית הספרות הבינאריות

ייצוג הקסה-דצימלי (בסיס 16)

- לדוגמה הייצוג הבינארי הבא:

11100011

- הוא המספר E3 בייצוג הקסה דצימלי
- בבסיס הקסה-דצימלי (0xE3)
- בהמרה מהקסה-דצימלי לעשרוני ניתן להשתמש בשיטת
- סכום המשקלים, כאשר משקלי הספרות הם:

(1, 16, 256 וכו')

שאלות?

רשימת האופרטורים

דוגמא	הסבר	משמעות אופרטור	סימן
2&3=2	ביט התוצאה יהיה 1 אם ורק אם שני הביטים המקבילים בפעולה שווים ל-1	AND	&
2|3=3	ביט התוצאה יהיה 1 אם לפחות אחד משני הביטים המקבילים בפעולה שווה ל-1	OR	|
2^3=1	ביט התוצאה יהיה 1 אם ורק אם בדיוק אחד משני הביטים המקבילים בפעולה שווה ל-1 (כלומר הביטים שונים זה מזה)	XOR	^

- ניתן להשתמש באופרטורים הבאים לקיצור: =&, =\|, =^

רשימת האופרטורים

דוגמא	הסבר	משמעות אופרטור	סימן
2<<1=4	מזיז את הביטים של משתנה המקור (האופרנד השמאלי) מספר צעדים שמאלה (עפ"י ערך האופרנד מימין). הפעולה מאפסת את התאים שנותרו ריקים מימין. הפעולה שקולה לכפל פי 2.	Shift Left	<<
2>>1=1	מזיז את הביטים של משתנה המקור (האופרנד השמאלי) מספר צעדים ימינה (עפ"י ערך האופרנד מימין). אלגוריתם מילוי התאים הריקים הוא תלוי מערכת. הפעולה שקולה לחלוקה פי 2.	Shift Right	>>
!2=253	מחליף את ערך של הביט: 1 והופך אותו ל-0 במקרה הפוך גורם ל- 0 להפוך ל-1	Complement	!

טבלאות פעולה

XOR

Bit 1	Bit 2	^ (xor)
0	0	0
0	1	1
1	0	1
1	1	0

OR

Bit 1	Bit 2	| (or)
0	0	0
0	1	1
1	0	1
1	1	1

AND

Bit 1	Bit 2	& (and)
0	0	0
0	1	0
1	0	0
1	1	1

שימוש במסכות (mask)

- נהוג לייצר מסכות שישמשו להדלקה או כיבוי של ביט:
  - יכולה להכיל 1 במיקום אותו נרצה להאיר
  - בעזרת שימוש באופרטור AND (&) בין המספר הרצוי והמסכה, נוכל לדעת מהו הביט הנמצא במספר במיקום הרצוי
- לדוגמה
  - 00001101 & 00001000 = 00001000
  - 00000101 & 00001000 = 00000000
- מכך נסיק:
  - אם התוצאה שהתקבלה היא <u>0</u>, סימן שהביט הרצוי היה 0
  - אם התוצאה שהתקבלה <u>שונה מ-0</u>, סימן שהביט הרצוי היה 1

דוגמה לפונקציה להדפסת ייצוג בינארי


```
void printBits(unsigned int number)
{
        unsigned int mask = 1<<31, result;  int i;
        for(i=0;i<32;i++)
        {
                result = number & mask;
                if(result)
                        putchar('1');
                else
                        putchar('0');
                mask >>=1;
        }
}
```

דוגמה לפונקציה להדפסת ייצוג עשרוני


```
void printNum(char mis[], int len)
{
        int i,sum=0,place=1;
        for(i=len-1;i>=0;i--)
        {
                if(mis[i] == '1')
                        sum+= (place);
        place*=2;
        }
 printf("\nThe number is: %d\n",sum);
}
```

דוגמה לתוכנית הראשית


```
int main()
{
   int num,len; char mis[32];
   printf("Enter integer number:\n");
   scanf("%d",&num);
   printBits(num);
   printf("Enter binary number:\n");
   scanf("%s",&mis);
   len=strlen(mis);
   printNum(mis,len);
   return 0;
}
```

תרגילים

תרגילים

1. כתוב פונקציה המקבלת מספר ומדפיסה את הייצוג הבינארי של המספר ולהיפך הבחירה איזו המרה תבוצע ע"י בחירת המפעיל.

2. כתוב פונקציה המקבלת מספר ומחזירה 1 אם הוא מתחלק ב-2 (0 אם לא). יש להשתמש בייצוג בינארי ובאופרטורים של ביטים.

3. כתוב פונקציה המקבלת מספר שלם, הפונקציה מעלה את המספר 2 בחזקת המספר השלם שנקלט.

עבור המספר 3 תחזיר התוכנית את התוצאה 8 , שהיא 2^3.

עבור המספר 5 הפונקציה תחזיר את המספר 32 שהוא 2^5

תרגילים

4. כתוב פונקציה המקבלת מספר שלם או תו ומחזירה כמה אחדות מופיעות בייצוג הבינארי שלו. יש לדאוג להתאים את הפונקציה לסוג המשתנה (unsigned int/char/int)
5. כתוב פונקציה המקבלת מספר בן 5 ספרות ומדפיסה את הייצוג ההקסה-דצימלי של המספר.
6. כתוב תוכנית המזיזה משתנה מסוג int 4 ביטים ימינה.

הדפס את המספר לפני ואחרי ההזזה. מה ערכי ארבעת הביטים השמאליים ביותר (שמולאו ע"י אלגוריתם ההזזה)?

בדוק את התשובה על משתנים signed/unsigned

פתרון תרגיל 4 לדוגמא


```
void  countOne(unsigned int num)
{
        int i, res=0;
        unsigned int mask=1, check;
        for(i=0;i<sizeof(int)*8;i++)
        {
                check= mask & num;
                if(check)
                    res++;
                mask<<=1;
        }
        printf("The number has  %d  1's in binary.\n",res);
}
```

תרגילים

7. כתוב תוכנית המקבלת 2 משתנים מסוג **unsigned int** ואורזת אותם בתוך **unsigned short** אחד.
כלומר, הבית השמאלי בתוצאה יהיה המספר הראשון והבית הימני יכיל את המספר השני.

8. כתוב את הפונקציה ההופכית, המפרקת מספר מסוג **short** ל-2 מספרים מסוג **char**. הפונקציה מקבלת 2 פרמטרים:

המספר הארוז ומספר נוסף המציין את מיקום המספר הרצוי (1 יסמן את המספר השמאלי ו-2 את המספר הימני).

הפונקציה תחזיר את המספר המבוקש מסוג **unsigned char** .

דוגמא אחרת לתוכנית המרה בין בסיסים


```
#include <stdio.h>
void main(void)
{
   char base_digits[16] =  {,'0' ,'1' ,'2' ,'3' ,'4' ,'5' ,'6' ,'7'
                                    ,'8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
   int converted_number[64];
   long int number_to_convert;
   int next_digit, base, index=0;

   /* get the number and base */
   printf("Enter number and desired base: ");
   scanf("%ld %i", &number_to_convert, &base);
```

דוגמא אחרת לתוכנית המרה בין בסיסים


```
/* convert to the indicated base */
  while (number_to_convert != 0)
 {
 converted_number[index] = number_to_convert % base;
 number_to_convert = number_to_convert / base;
 ++index;
 }
  /* now print the result in reverse order */
  --index;  /* back up to last entry in the array */
  printf("\nConverted Number = ");
  for(  ; index>=0; index--) /* go backward through array */
  {
 printf("%c", base_digits[converted_number[index]]);
  }
  printf("\n");
}
```

תרגילים

9. לפניך פונקציה המבצעת משימה מסוימת. עליך לעבור על הפונקציה בהנחה כי המספרים שנקטו בראש הפונקציה הם X=10 ו- Y=6 ולרשום מה יודפס בסופה.

```
#include <stdio.h>
void Print(int x, int d)
{
        char buffer[33];
        int i=0;
        for (;d>0;d--)
        {
                buffer[i++] = '0'+ (x & 1);
                x >>= 1;
        }
        while(index >0 )
                printf("%c",buffer[--i]);
}
```

תרגילים

10. לפניך תוכנית הכוללת פונקציה המבצעת משימה מסוימת. עליך לעבור על התוכנית ולרשום מה יודפס בסופה.

```
int mystery(unsigned char bits)
{
        int total=0, i;
        for(i=1;i<=bits;i++){
                bits<<=2;
                total++;{
        return (total);
}
int main()
{
        unsigned int number=8;
        printf("The answer is: %d\n", mystery(number<<1));
}
```

סיביות וביטים - סיכום

- מדוע ביטים? מהם סיביות? משמעות ייצוג מידע
- ייצוג מידע בינארי / הקסה-דצימלי / אוקטאלי
  - ייצוג של בתים
    - מספרים
    - תווים ומחרוזות
    - פקודות (instructions)
- פעולות ברמת הביטים
  - לוגיקה בוליאנית, יחסי or, and, not, xor
  - הזזות ימינה ושמאלה של ביטים והמשמעות של זה.
  - כיצד נכתוב זאת בתוכניות C ?

שאלות?