גירסה 1.01 – 25.03.2010
גירסה 1.00 – 24.09.2004

שפת C

אופרטורים הפועלים על סיביות

ניר אדר

Nir Adar
Email: nir@underwar.co.il
Home Page: http://www.underwar.co.il

אנא שלחו תיקונים והערות אל המחבר.

פתיחה ואופרטורים בסיסיים

אופרטורים הפועלים על סיביות מתייחסים לצורה בה אנחנו מייצגים נתונים במחשב.
נתאר קצת את זיכרון המחשב: היחידה הבסיסית ששומרת במחשב מידע מכונת ביט (Bit). ביט היא יחידה היכולה לקבל רק שני ערכים: 0 או 1, דלוק או מכובה. קבוצה של 8 ביטים נקראת בית (Byte).
כיצד נשמור בזיכרון נתונים אחרים פרט לאפסים ואחדות, למשל - מספרים, אותיות וכו'?
התשובה: בעזרת ייצוג של מספרים ואותיות כרצפים מוסכמים של אפסים ואחדות, ופיענוח של האפסים והאחדות על ידי המחשב.

ייצוג המספרים בבסיס בינארי הוא הדרך בה אנחנו שומרים מספרים שלמים במחשב.
אני מניח שהקורא מכיר את הבסיס הבינארי, ולכן לא אכלול תאור שלו במסמך זה.

בעוד שהפעולות הרגילות בשפת C, כגון חיבור, חיסור וכו' מתייחסים אל המספר כאל המספר עצמו, הפעולות הבינאריות מתייחסות אל הסיביות השונות המרכיבות את המספר.

הפעולות הבסיסיות שנוכל לבצע בין שני מספרים בינאריים הינם AND ו-OR.
AND בין שני מספרים בינאריים משמעותו "התוצאה של ה-AND תהיה 1 אם גם המספר הראשון וגם השני הם 1".
OR בין שני מספרים בינאריים משמעותו "התוצאה של ה-AND תהיה 1 המספר הראשון ו/או המספר השני הם 1".

הטבלה הבאה תסכם זאת. עבור שני מספרים בינאריים, A ו-B, נראה את התוצאה של ביצוע AND ושל ביצוע OR. טבלאות כגון זו מכונות "טבלאות אמת":

A	B	A AND B	A OR B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

האופרטור AND ממומש בשפת C על ידי & והאופרטור OR על ידי |.

בשפת C כאשר אנחנו מפעילים את אחד האופרטורים המבצעים פעולות על ביטים על משתנה, הפעולה מבוצעת על כל אחד מהביטים של המשתנה. לדוגמא בקוד הבא:

```
int x1, x2;
int res = x1 & x2;
```

יבוצע AND של הביטים של x1 מול אלו של x2, אחד מול השני.

לדוגמא:

ניקח את המספרים $x_1 = 234$ ו-$x_2 = 63$. הייצוג שלהם בבינארית הינו: $x_1 = 11101010_2$ ו-$x_2 = 111111_2$. (הסימון ב-2 תחתון מציין כי המספר הינו מספר בינארי).

פעולת AND בין המספרים תבוצע כך:

```
11101010
111111
=
101010
```

תוצאת ה-AND תהיה 42.

XOR

בהינתן שני מספרים A ו-B, האופרטור XOR משמעותו: XOR יחזיר 1 אם A הוא 1, או B הוא 1, אבל לא שניהם 1. טבלת האמת של XOR נתונה להלן:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

אופרטור ה-XOR בשפת C מיוצג על ידי ^. לדוגמא:

```
int x1, x2;
int res = x1 ^ x2;
```

גם אופרטור זה בשפת C מבצע את פעולת ה-XOR סיבית אחרי סיבית, עבור כל אחת מהסיביות ב-x1 מול x2.

אופרטור המשלים

האופרטור ~ הינו המשלים של המספר. אופרטור זה הוא אופרטור אונרי (פועל על משתנה יחיד).
בהינתן מספר בעל 8 בתים, למשל, המשלים של x, שנסמנו x' הוא איבר המקיים
$x + x' = 0\text{xFFFFFFFF}$.
לדוגמא:

```
~0x00000008 == 0xFFFFFFF7
~0xFFFFFFF8 == 0x00000007
```

אופרטורים של הזזה

האופרטורים האחרונים הינם אופרטורים של הזזה. קיימת הזזה שמאלה והזזה ימינה.
המשמעות של **הזזה שמאלה** היא לקחת את המספר הנתון, ולהוסיף לו אפסים מימין.
לדוגמא, יהיה המספר הבינארי 11101_2. הזזה שלו פעמיים לכיוון שמאל הינה: 1110100_2.
פעולה מתמטית שאנחנו מקבלים כתוצאה מתכונות המספרים הבינאריים היא כפל: הזזה של מספר לשמאל הינה הכפלה של המספר. לדוגמא: המספר 100_2 הינו המספר 4 בבסיס עשרוני, ואילו המספר 1000_2 הינו 8 בבסיס עשרוני. באופן דומה, הזזת מספר 3 סיביות שמאלה הינה הכפלתו ב-2^3.
הזזה לימין, משמעותה זריקת האיברים הראשונים מצד ימין: עבור המספר 11101_2, הזזה של שני מספרים ימינה תיתן את התוצאה: 111_2. ניסוח זה אינו מדויק, אלא מופשט קצת ונועד להבנה אינטואיטיבית של האופרטור.

כדי לנסח במדויק את המשמעות של הזזה לימין נזכר כי לכל מספר המיוצג במחשב מספר קבוע של ביטים. נניח כעת שכל מספר שאנחנו מתעסקים עימו מיוצג על ידי 8 ביטים. הביט השמאלי ביותר יקרא ה-MSB והביט הימני ביותר יקרא ה-LSB. ביצוע הזזה (shift) ימינה משמעותו זריקת הערך הימני ביותר, ושכפול של הערך השמאלי ביותר.

לדוגמא, עבור המספר 00110011_2 הזזה ימינה פעם אחת תיתן את המספר 00011001_2.

לעומת זאת, המספר 10110011_2 יהפוך ל-11011001_2 אם נזיז אותו פעם אחת ימינה (1 נעלם מצד ימין, וה-1 השמאלי שוכפל).

הזזה ימינה אינה חלוקה מדויקת ב-2, תיאור מדויק יותר שלה יהיה שהיא חלוקה ב-2 ללא שארית.

האופרטור הזזה שמאלית מיוצג ב-C על ידי << ואילו האופרטור הזזה ימנית מיוצג על ידי >>.

דוגמא:

```
#include <stdio.h>

int main()
{
	int i = 1;
	int j = i << 5;
	printf("%d\n", j);

}
```

המשמעות של i << 5 הינה להזיז את הביטים ב-i 5 פעמים שמאלה. $i = 1_2$ ולכן על ידי 5 הזזות יתקיים כי $j = 100000_2$ ועל המסך יודפס הערך 32.

שימושים לאופרטורים

למה בעצם אנחנו צריכים את האופרטורים הפועלים על ביטים בשפת C? נציג כעת מספר שימושים ורעיונות לשימושים לאופרטורים בשפה.

כפי שכבר ראינו, ניתן לממש הכפלה יעילה בחזקה של 2 על ידי שימוש באופרטורים של הזזה. שימוש באופרטורים זה הוא יעיל במיוחד מכיוון שהוא בד"כ נתמך על ידי החומרה.

טריק נוסף הוא שימוש ב-XOR לאיפוס מספר. נשים לב ש-XOR של מספר עם עצמו, יתן תמיד 0, ולכן הפקודה הבאה תאפס את המשתנה:

```
int x = 4;
x ^= x;
```

בשפות אסמבלר שונות טריק זה הוא נפוץ ביותר.

האופרטורים הבינריים עוזרים במקרים שונים. נניח למשל שנרצה להעביר לפונקציה מסוימת פרמטר המייצג אפשרויות. למשל, נניח שאנחנו כותבים פונקציה הפותחת קובץ. הפונקציה יכולה לפתוח את הקובץ לכתיבה או לקריאה. הפונקציה יכולה ליצור קובץ חדש אם הקובץ לא קיים או להכשל אם הקובץ קיים. יתכנו גם שילוב בין האפשרויות. כעת, אפשרות אחרת היא לכתוב את הצהרת הפונקציה כך:

```
enum BOOL { FALSE, TRUE };
typedef enum BOOL BOOL;

FILE* file_open(char* file_name, BOOL isRead, BOOL isWrite, BOOL
createNew);
```

כשקריאה לפונקציה, תראה כך, למשל:

```
file_open("myfile.txt", TRUE, FALSE, FALSE);
```

דרך נוחה יותר, היא להעביר משתנה יחיד. נקבע כי הביט הראשון שלו יהיה 1 אם הקובץ לקריאה ו-0 אם לא. הביט השני יהיה 1 אם הקובץ לכתיבה ו-0 אחרת, והביט השלישי יהיה 1 אם יוצרים קובץ חדש במקרה הצורך, ו-0 אחרת.

נגדיר את הקבועים המתאימים:

```
#define READ      1
#define WRITE     2
#define CREATE    4
```

וכעת הצהרת הפונקציה תראה כך:

```
FILE* file_open(char* file_name, int options);
```

הקריאה לפונקציה לעומת זאת יכולה להראות כך:

```
file_open("myfile.txt", READ);
```

בנוסף, על ידי האופרטור OR נוכל לשלב כמה אפשרויות:

```
file_open("myfile.txt", WRITE | CREATE);
```

עד כאן טוב ויפה, אבל איך file_open() תמומש במקרה זה? איך נבדוק את הביט המתאים ונראה האם פרמטר הועבר אלינו? התשובה היא שנשתמש באופרטור AND, לדוגמא:

```
FILE* file_open(char* file_name, int options)
{
	...
	if (options & WRITE != 0)
	{
		/* Write flag is up… */
	}
	...
}
```

EOF