

Η ΓΛΩΣΣΑ C

Μάθημα 22:

Πράξεις bit

Δημήτρης Ψούνης



www.psounis.gr



Περιεχόμενα Μαθήματος

A. Bits & Bytes

1. Εισαγωγή

1. Αποδομώντας μία Μεταβλητή
2. Συνάρτηση εξαγωγής bit

2. Τελεστές Ολίσθησης

1. Αριστερή ολίσθηση
2. Δεξιά ολίσθηση

3. Λογικοί Τελεστές (bitwise)

1. Λογικό ΚΑΙ (&)
2. Λογικό Ή (|)
3. Λογικό Αποκλειστικό Ή (^)
4. Λογικό ΟΧΙ (~)
5. Συνάρτηση set_bit

4. Δομές και μεταβλητές bit

1. Συντακτικό bit σε δομές.
2. Παράδειγμα

B. Ασκήσεις



A. Bits & Bytes

1. Εισαγωγή

1. Αποδομώντας μία μεταβλητή

- Γνωρίζουμε ότι οι τύποι δεδομένων στην C δεσμεύουν κάποιο προκαθορισμένο πλήθος bytes.
- Όπως είδαμε στο μάθημα 3 «Μεταβλητές και Σταθερές»
 - Ένας char ακέραιος δεσμεύει 1 byte
 - Ένας short ακέραιος δεσμεύει 2 bytes
 - Ένας int ακέραιος συνήθως δεσμεύει 4 bytes
 - Ένας long ακέραιος δεσμεύει 8 bytes
- Και γνωρίζουμε ότι κάθε ακέραιος έχει δύο εκδοχές
 - Την προσημασμένη (μπορούμε να αποθηκεύσουμε αρνητικές και θετικές τιμές)
 - Την μη προσημασμένη (μπορούμε να αποθηκεύσουμε μόνο θετικές τιμές)
- Υπενθυμίζουμε (επόμενη διαφάνεια) και τα εύρη τιμών των τύπων αυτών



A. Bits & Bytes

1. Εισαγωγή

1. Αποδομώντας μία μεταβλητή

Όνομα Τύπου Δεδομένων	Συμβολισμός	bytes	Εύρος τιμών
Χαρακτήρας	char	1	-128 έως 127
Μικρός Ακέραιος	short	2	-32768 έως 32767
Ακέραιος	int	4	-2147483648 έως 2147438647
Μεγάλος Ακέραιος	long	4	-2147483648 έως 2147438647

Όνομα Τύπου Δεδομένων	Συμβολισμός	bytes	Εύρος τιμών
Μη προσημασμένος Χαρακτήρας	unsigned char	1	0 έως 255
Μη προσημασμένος Μικρός Ακέραιος	unsigned short	2	0 έως 65535
Μη προσημασμένος Ακέραιος	unsigned int	4	0 έως 4294967295
Μη προσημασμένος Μεγάλος Ακέραιος	unsigned long	4	0 έως 4294967295

A. Bits & Bytes

1. Εισαγωγή

1. Αποδομώντας μία μεταβλητή

- Γνωρίζουμε ότι:
 - Κάθε byte έχει 8 bits
 - Κάθε αριθμός αποθηκεύεται στο δυαδικό σύστημα αρίθμησης.
- Οπότε μπορούμε να βρούμε τη δυαδική αναπαράσταση ενός ακεραίου, αποδομώντας τον αριθμό στα αντίστοιχα bits.
- Στο πρόγραμμα της επόμενης διαφάνειας, αποδομούμε έναν unsigned char (ο απλούστερος μη προσημασμένος ακέραιος) στα αντίστοιχα bits τα οποία περιέχει:

A. Bits & Bytes

1. Εισαγωγή

1. Αποδομώντας μία μεταβλητή

```
/* bits_in_char.c Bits σε έναν μη προσημασμένο χαρακτήρα */

#include <stdio.h>

#define BITS 8

int main()
{
    unsigned char byte = 3;
    int bits[BITS];
    int i;

    for (i=0; i<BITS; i++)
    {
        bits[BITS-1-i]=byte%2;
        byte=byte/2;
    }

    for (i=0; i<BITS; i++)
        printf("%d", bits[i]);

    return 0;
}
```

A. Bits & Bytes

1. Εισαγωγή

1. Αποδομώντας μία μεταβλητή

- Δεδομένου του unsigned char byte (π.χ. 3)
- Η δυαδική του αναπαράσταση είναι:

0 0 0 0 0 0 1 1

- Η πράξη (υπόλοιπο διαίρεσης με το 2):

byte%2

- απομονώνει το τελευταίο bit.
- ενώ η πράξη (ακέραια διαίρεση με το 2):

byte/2

- μετακινεί τα bits του αριθμού μία θέση δεξιά:

0 0 0 0 0 0 0 1

Για υπενθύμιση στην δυαδική αναπαράσταση αριθμών, βλέπε «Εισαγωγή στους Η/Υ – Συστήματα Αρίθμησης»

A. Bits & Bytes

1. Εισαγωγή

1. Αποδομώντας μία μεταβλητή

- Με βάση τα παραπάνω, η επανάληψη αυτών των πράξεων απομονώνει ένα – ένα τα bits του αριθμού.
 - Σύμφωνα με τον αλγόριθμο μετατροπής ενός δεκαδικού στον αντίστοιχο δυαδικό.
- Π.χ. για byte=43

byte=	0	0	1	0	1	0	1	1	byte%2=1	byte=byte/2=21
byte=	0	0	0	1	0	1	0	1	byte%2=1	byte=byte/2=10
byte=	0	0	0	0	1	0	1	0	byte%2=0	byte=byte/2=5
byte=	0	0	0	0	0	1	0	1	byte%2=1	byte=byte/2=2
byte=	0	0	0	0	0	0	1	0	byte%2=0	byte=byte/2=1
byte=	0	0	0	0	0	0	0	1	byte%2=1	byte=byte/2=0
byte=	0	0	0	0	0	0	0	0	byte%2=0	byte=byte/2=0
byte=	0	0	0	0	0	0	0	0	byte%2=0	byte=byte/2=0

- Και αποθηκεύουμε τα bytes με αντίστροφη σειρά εξαγωγής: $(43)_{10} = (00101011)_2$

A. Bits & Bytes

1. Εισαγωγή

2. Συνάρτηση εξαγωγής του bit σε μία θέση

- Γενικεύοντας το συλλογισμό μπορούμε να κατασκευάσουμε τις εξής συναρτήσεις:
- Τη συνάρτηση `get_bit` που επιστρέφει το bit που είναι στην i-οστή θέση του byte:

```
int get_bit(int pos, unsigned char byte_char)
{
    int i;

    for (i=0; i<BITS-pos-1; i++)
        byte_char=byte_char/2;
    return byte_char%2;
}
```

- Και τη συνάρτηση εκτύπωσης που τυπώνει ένα byte

```
void bits_print(unsigned char byte)
{
    int i;
    for (i=0; i<BITS; i++)
        printf("%d", get_bit(i, byte));
}
```

Ολοκληρωμένο το πρόγραμμα είναι το `bits_get_function.c`

A. Bits & Bytes

2. Τελεστές Ολίσθησης

1. Αριστερή ολίσθηση

- Με τον όρο αριστερή ολίσθηση (left shift), εννοούμε:
 - Την μετακίνηση μία θέση αριστερά των bits
 - Το αριστερότερο (καλείται και πιο σημαντικό) bit χάνεται.
 - Το δεξιότερο bit (καλείται και λιγότερο σημαντικό) γίνεται 0.
- Παράδειγμα:
 - Αρχικό byte:

0	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---
 - Μετά από αριστερή ολίσθηση

0	1	0	1	0	1	1	0
---	---	---	---	---	---	---	---
- Παρατηρήσεις:
 - Ισοδυναμεί με πολλαπλασιασμό του αριθμού που απεικονίζεται επί δύο
 - Και ακόμη ακριβέστερα (αν B είναι το byte) με την πράξη: $B = (2 * B) \% 2^8$
- Ο τελεστής αριστερής ολίσθησης στη C είναι ο `<<`
- και χρησιμοποιείται ως εξής:

```
var << n
```

- όπου var μια μεταβλητή (π.χ. unsigned char)
- και n πόσες αριστερές ολισθήσεις θα γίνουν.

A. Bits & Bytes

2. Τελεστές Ολίσθησης

2. Δεξιά ολίσθηση

- Με τον όρο δεξιά ολίσθηση (right shift), εννοούμε:
 - Την μετακίνηση μία θέση δεξιά των bits
 - Το δεξιότερο bit (καλείται και λιγότερο σημαντικό) χάνεται.
 - Το αριστερότερο bit (καλείται και πιο σημαντικό) γίνεται 0.
- Παράδειγμα:
 - Αρχικό byte:

0	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---
 - Μετά από δεξιά ολίσθηση

0	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---
- Παρατηρήσεις:
 - Ισοδυναμεί με ακέραια διαίρεση του αριθμού που απεικονίζεται επί δύο
- Ο τελεστής δεξιάς ολίσθησης στη C είναι ο `>>`
- και χρησιμοποιείται ως εξής:

```
var >> n
```

- όπου var μια μεταβλητή (π.χ. unsigned char)
- και n πόσες δεξιές ολισθήσεις θα γίνουν.

A. Bits & Bytes

2. Τελεστές Ολίσθησης

2. Δεξιά ολίσθηση

- Με χρήση της αριστερής ολίσθησης μπορούμε να απλοποιήσουμε τη συνάρτηση εξαγωγής bit ως εξής:

```
int get_bit(int pos, unsigned char byte)
{
    return (byte >> BITS-1-pos)%2;
}
```

- Π.χ. αν θέλουμε να πάρουμε την 2^η θέση (pos=2):

0	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---
- Πρέπει να κάνουμε 5 ολισθήσεις (BITS-1-pos)

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---
- και στη συνέχεια κάνοντας mod 2 παίρνουμε το τελευταίο bit

Ολοκληρωμένο το πρόγραμμα είναι το `bits_get_function_shift.c`

- Οι τελεστές `<<` και `>>` δουλεύουν μόνο με ακραίους: short, int, long και όχι float ή double

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

1. Λογικό ΚΑΙ (&)

- Ο τελεστής λογικό ΚΑΙ (&) εκτελεί τη λογική πράξη AND μεταξύ των αντιστοίχων bits δύο αριθμών:

X	Y	AND
0	0	0
0	1	0
1	0	0
1	1	1

- Παράδειγμα:

0 0 1 0 1 0 1 1 (43)

& 0 1 1 0 0 1 1 1 (103)

0 0 1 0 0 0 1 1 (35)

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

1. Λογικό ΚΑΙ (&)

- Το λογικό ΚΑΙ μπορεί να χρησιμοποιηθεί για να κάνουμε την ονομαζόμενη «μάσκα» και να απομονώσουμε ένα bit ενός αριθμού.
- Π.χ. αν θέλουμε να πάρουμε την 2^η θέση (pos=2):

1 0 1 0 1 0 1 1

- Πρέπει να κάνουμε 5 ολισθήσεις (BITS-1-pos)

0 0 0 0 0 1 0 1

- και στη συνέχεια κάνοντας μάσκα με το 1

0 0 0 0 0 0 0 1

- παίρνουμε το τελευταίο bit

0 0 0 0 0 0 0 1

- Συνεπώς αλλάζουμε την συνάρτηση ως εξής:

```
int get_bit(int pos, unsigned char byte)
{
    return (byte >> BITS-1-pos) & 1;
}
```

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

2. Λογικό Ή (|)

- Ο τελεστής λογικό ΚΑΙ (&) εκτελεί τη λογική πράξη AND μεταξύ των αντιστοίχων bits δύο αριθμών:

X	Y	OR
0	0	0
0	1	1
1	0	1
1	1	1

- Παράδειγμα:

0 0 1 0 1 0 1 1 (43)

| 0 1 1 0 0 1 1 1 (103)

0 1 1 0 1 1 1 1 (111)

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

2. Λογικό Ή (|)

- Το λογικό Ή μπορεί να χρησιμοποιηθεί για να θέσουμε ένα bit σε 1
- Π.χ. για να αλλάξουμε το 2ο bit

1 0 0 0 1 0 1 1

- Ξεκινάμε μάσκα με το 1

0 0 0 0 0 0 0 1

- Το φέρνουμε στη σωστή θέση (BITS-1-POS) αριστερές ολισθήσεις

0 0 1 0 0 0 0 0

- Και το λογικό Ή θα δώσει το αποτέλεσμα που θέλουμε:

1 0 1 0 1 0 1 1

- Γράφουμε μερικώς τη συνάρτηση:

```
void set_bit(int pos, unsigned char *byte, int val)
{
    unsigned char mask = 1;
    if (val==1) *byte = (mask << BITS-1-pos) | *byte;
}
```

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

3. Λογικό Αποκλειστικό Ή (^)

- Ο τελεστής λογικό ΑΠΟΚΛΕΙΣΤΙΚΟ Ή (^) εκτελεί τη λογική πράξη XOR μεταξύ των αντιστοιχών bits δύο αριθμών:

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

- Παράδειγμα:

0 0 1 0 1 0 1 1 (43)

^ 0 1 1 0 0 1 1 1 (103)

0 1 0 0 1 1 0 0 (76)

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

4. Λογικό ΟΧΙ (~)

- Ο τελεστής λογικό ΟΧΙ (~) εκτελεί τη λογική πράξη NOT επί ενός αριθμού:

X	XOR
0	1
1	0

- Παράδειγμα:

~ 0 0 1 0 1 0 1 1 (43)

1 1 0 1 0 1 0 0 (212)

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

4. Λογικό ΟΧΙ (~)

- Με το NOT μπορούμε να κατασκευάσουμε ενδιαφέρουσα μάσκα για να θέσουμε ένα bit σε 0.
- Π.χ. για να αλλάξουμε το 2ο bit

1 0 1 0 1 0 1 1

- Ξεκινάμε τη μάσκα με το 1

0 0 0 0 0 0 0 1

- Το φέρνουμε στη σωστή θέση (BITS-1-POS) αριστερές ολοσθήσεις

0 0 1 0 0 0 0 0

- Αντιστρέφουμε τη μάσκα:

1 1 0 1 1 1 1 1

- Και κάνοντας λογικό ΚΑΙ με το αρχικό byte κάναμε το επιθυμητό:

1 0 0 0 1 0 1 1

- Σε κώδικα:

```
*byte = ~(mask << BITS-1-pos) & *byte;
```

A. Bits & Bytes

3. Λογικοί Τελεστές (bitwise)

5. Συνάρτηση set_bit

- Συνεπώς κατασκευάσαμε τη συνάρτηση:

```
void set_bit(unsigned char *byte, int pos, int val)
{
    unsigned char mask = 1;

    if (val==0)
        *byte = ~(mask << BITS-1-pos) & *byte;
    else
        *byte = (mask << BITS-1-pos) | *byte;
}
```

- Μέσω της οποίας μπορούμε να κάνουμε ενέργειες όπως:

```
int main()
{
    unsigned char byte = 53;
    set_bit(&byte,2,1);

    return 0;
}
```

Ολοκληρωμένο το πρόγραμμα είναι το bits_set_function.c



A. Bits & Bytes

4. Δομές και μεταβλητές bit

1. Συντακτικό bit σε δομές.

- Στις δομές (structs) μπορούμε να ορίσουμε πεδία που είναι bits.
- Γίνεται με το εξής συντακτικό:

```
struct name
{
    type bit_name1: n1;
    type bit_name2: n2;
    ...
}
```

- όπου ορίζουμε πρώτα:
 - Σε τι μεταβλητές θα μεταφράζονται τα bits (είναι int ή unsigned int)
- έπειτα δίνουμε ένα όνομα
- και μετά την άνω κάτω τελεία
 - ορίζουμε το πλήθος των bits που θα χρησιμοποιήσει.
- Μπορούμε να χρησιμοποιήσουμε όσα τέτοια πεδία θέλουμε σε μία δομή.



A. Bits & Bytes

4. Δομές και μεταβλητές bit

2. Παράδειγμα

- Για παράδειγμα έστω ότι κατασκευάζουμε μία δομή «Φοιτητής» η οποία:
 - Έχει ένα πεδίο «ενεργός» που είναι 0 ή 1
 - Έχει ένα πεδίο «είδος φοίτησης» που είναι 0 (προπτυχιακός), 1(μεταπτυχιακός), 2(διδακτορικός)
 - και συνηθισμένα πεδία όπως όνομα, επώνυμο κ.λπ.
- Ορίζουμε τη δομή με τον εξής τρόπο:

```
struct student
{
    unsigned int active: 1;
    unsigned int type: 2;
    char name[80];
    ...
}
```

- Στην μεταβλητή active μπορούμε να βάζουμε τιμή 0 ή 1
- Στην μεταβλητή type μπορούμε να βάζουμε τιμή 0 ή 1 ή 2
 - Αφού χρησιμοποιούμε 2 bits μπορούμε να αποθηκεύουμε 4 αριθμούς (0,1,2,3)



A. Bits & Bytes

4. Δομές και μεταβλητές bit

2. Παράδειγμα

```
/* bits_in_struct.c Δομή που περιέχει πεδία bit */
#include <stdio.h>
#include <string.h>

struct student
{
    unsigned int active: 1;
    unsigned int type: 2;
    char name[80];
};

main()
{
    struct student bob;

    bob.active=0;
    bob.type=2;
    strcpy(bob.name, "Bob");

    printf("%d %d %s", bob.active, bob.type, bob.name);
}
```



A. Bits & Bytes

5. Συναρτήσεις διαχείρισης μνήμης

1. Η συνάρτηση memset

- Προσφέρονται 3 συναρτήσεις οι οποίες διαχειρίζονται απ'ευθείας bytes:
- Η συνάρτηση memset:


```
void *memset(void *dest, int c, size_t count);
```

 - και έχει οριστεί στο `stdio.h`
 - Η συνάρτηση θέτει στα πρώτα count byte του dest το χαρακτήρα c.
- Παράδειγμα: Αν str="xxxxxxxx", η εντολή:


```
memset(str, 'a', 4)
```

 - θα μετατρέψει το str="aaaaxxxxxx"

A. Bits & Bytes

5. Συναρτήσεις διαχείρισης μνήμης

2. Η συνάρτηση memcpy

- Η συνάρτηση memcpy:

```
void *memcpy(void *dest, void *src, size_t count)
```

- και έχει οριστεί στο `stdio.h`
- Η συνάρτηση αντιγράφει count bytes από το src στο dest.
- Αν οι δύο χώροι μνήμης επικαλύπτονται ενδέχεται να αποτύχει.
- Παράδειγμα: Αν η str1="xxxxxxxx" και η str2="aaaa", τότε η εντολή:

```
memset(str1, str2, 3)
```

 - θα μετατρέψει το str1="aaaxxxxxxx"
- Παράδειγμα επικαλυπτόμενης μνήμης που η memcpy μπορεί να αποτύχει (π.χ. για count=4):

a	x	x	x	x	a	a	a
str1	str2						

A. Bits & Bytes

5. Συναρτήσεις διαχείρισης μνήμης

2. Η συνάρτηση memmove

- Η συνάρτηση memmove:

```
void *memmove(void *dest, void *src, size_t count)
```

- και έχει οριστεί στο `stdio.h`
- Η συνάρτηση αντιγράφει count bytes από το src στο dest.
- Αν οι δύο χώροι μνήμης επικαλύπτονται κάνει την αναμενόμενη ενέργεια. Στο προηγούμενο παράδειγμα:

a	x	x	x	x	a	a	a
str1	str2						

- Η εντολή

```
memset(str1, str2, 4)
```
- Θα δώσει τη συμβολοσειρά

x	x	x	a	x	a	a	a
str1	str2						

B. Ασκήσεις

1. LSB & MSB

Γράψτε μία συνάρτηση int LSB(unsigned char byte) η οποία

- Θα επιστρέφει το λιγότερο σημαντικό ψηφίο (δεξιότερο bit) ενός byte

Γράψτε μία συνάρτηση int MSB(unsigned char byte) η οποία

- Θα επιστρέφει το πιο σημαντικό ψηφίο (αριστερότερο bit) ενός byte

Επαληθεύστε την ορθότητα των δύο συναρτήσεων μέσω κατάλληλης συνάρτησης main

B. Ασκήσεις

2. Δεξιά περιστροφή

Γράψτε μία συνάρτηση void rotate(unsigned char *byte) η οποία

- Θα δέχεται ένα byte και θα μετακινεί κυκλικά κατά μία θέση δεξιά όλα τα bit του αριθμού
- Π.χ. το 00001111 θα πρέπει να μετατραπεί με μία δεξιά κυκλική ολίσθηση στο 10000111.

Επαληθεύστε την ορθότητα των δύο συναρτήσεων μέσω κατάλληλης συνάρτησης main

B. Ασκήσεις

3. Αρχείο Εγγραφών

Η δομή σημείο περιέχει τα εξής στοιχεία:

- Αν είναι ορατό (0 ή 1)
- Το χρώμα του (από 0 έως 63)
- Τις συντεταγμένες του στον 2Δ χώρο (int)

Κατασκευάστε ένα πρόγραμμα το οποίο:

- Θα δημιουργεί ένα αρχείο με την εξής γραμμογράφιση:
 - Αρχικά θα υπάρχει το πλήθος των εγγραφών του
 - Έπειτα θα ακολουθούν 10.000 εγγραφές τυχαίων σημείων σε δυαδική μορφή.
- Οι δύο πρώτες μεταβλητές (ορατότητα και χρώμα) θα πρέπει να λαμβάνουν όσο το δυνατόν λιγότερο χώρο (bits)

Επαληθεύστε ότι η εγγραφή έχει γίνει σωστά, διαβάζοντας το αρχείο που κατασκευάσατε.