

ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Χαμηλού Επιπέδου Προγραμματισμός II (Κεφάλαια 20.2, ΚΝΚ-2ΕΔ)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

<http://www.cs.ucy.ac.cy/courses/EPL232>

**Το μάθημα αυτό δομήθηκε βάση των διαλέξεων του
Αναπλ. Καθηγητή Δημήτρη Ζεϊναλιπούρ**



Περιεχόμενο Διάλεξης

- Εισαγωγή
- Δυαδικοί Τελεστές (Bitwise Operators)
 - Τελεστές Ολίσθησης (Shift): \gg , \ll
 - Τελεστές Συμπλήρωμα (\sim), AND ($\&$), OR (\mid), XOR (\wedge)
 - Χρήση Τελεστών για πρόσβαση σε bit
 - Ανάθεση, Διαγραφή και Δοκιμή Bit
 - Ονόματα σε Μάσκες
 - Παράδειγμα Κρυπτογράφησης XOR
- Δομές με Δυφία (Struct with bit-fields)
- Θέματα Ευθυγράμμισης Μνήμης με Δομές Διφυιών

Διάλεξη #18



Χρήση των τελεστών bitwise για την πρόσβαση σε δυαδικά πεδία

- Η επεξεργασία μιας ομάδας από διαδοχικά bits (**ένα πεδίο από bits**) είναι πιο περίπλοκη από την επεξεργασία μονών bits.
- Κοινές λειτουργίες σε πεδίο bit:
 - Τροποποίηση ενός πεδίου bit
 - Ανάκτηση ενός πεδίου bit

Τροποποίηση ενός δυαδικού πεδίου

- Η τροποποίηση ενός πεδίου bit απαιτεί δύο λειτουργίες:
 - Το bitwise **and** (για να καθарίσετε το πεδίο bit)
 - Το bitwise **or** (για την αποθήκευση νέων bits στο πεδίο bit)

- Παράδειγμα:

```
i = i & ~0x0070 | 0x0050;  
/* αποθηκεύει 101 στα bits 4-6 */
```

- Ο τελεστής & καθарίζει τα bits 4–6 του i; ο τελεστής | θέτει τα bits 6 και 4.

Τροποποίηση ενός δυαδικού πεδίου

- Για να γενικεύσουμε το παράδειγμα, ας υποθέσουμε ότι το j περιέχει την τιμή που πρέπει να αποθηκευτεί στα bits 4 – 6 του i .
- Το j θα πρέπει να μετατοπιστεί σε θέση πριν από το bitwise :

```
i = (i & ~0x0070) | (j << 4);  
/* αποθηκεύει το j στα bits 4-6 */
```

- Δεδομένου πως ο τελεστής $|$ έχει χαμηλότερη προτεραιότητα από τους $\&$ και \ll , οι παρενθέσεις μπορούν να αποσυρθούν:

```
i = i & ~0x0070 | j << 4;
```



Ανάκτηση ενός δυαδικού πεδίου

- Η λήψη ενός πεδίου bit στο δεξιό άκρο ενός αριθμού (στα λιγότερο σημαντικά bit) είναι εύκολη:

```
j = i & 0x0007;  
/* ανακτά τα bits 0-2 */
```

- Αν το πεδίο bit δεν είναι στο δεξιό άκρο του i , μπορούμε πρώτα να μετατοπίσουμε το πεδίο bit στο τέλος πριν από την ανάκτηση και λήψη του πεδίου με τον τελεστή $\&$:

```
j = (i >> 4) & 0x0007;  
/* ανακτά τα bits 4-6 */
```



Παράδειγμα (Κρυπτογράφηση XOR)

- Ο Αλγόριθμος **Συμμετρικής Κρυπτογράφησης XOR** (από και προς με το ίδιο ή παρόμοιο κλειδί) είναι η απλούστερη μέθοδος για κωδικοποίηση ενός κειμένου
 - Στηρίζεται στην ακόλουθη λογική:

Κρυπτογράφηση

01111010 (ASCII #122 'z') => Χαρακτήρας

00100110 (ASCII #38 '&') => **Κλειδί (Shared Key)**

^-----

01011100 (ASCII #92 '\') => Κρυπτογραφημένος Χαρακτήρας

Αποκρυπτογράφηση

01011100 (ASCII #92 '\') => Κρυπτογραφημένος Χαρακτήρας

00100110 (ASCII #38 '&') => **Κλειδί (Shared Key)**

^-----

01111010 (ASCII #122 'z') => Αρχικός Χαρακτήρας

Παράδειγμα (Κρυπτογράφηση XOR)

- Έστω ένα δείγμα αρχείου με όνομα `msg`:

```
Trust not him with your secrets, who, when left  
alone in your room, turns over your papers.
```

--Johann Kaspar Lavater (1741-1801)

- Μια εντολή που κρυπτογραφεί το `msg`, και αποθηκεύει το κρυπτογραφημένο μήνυμα στο `newmsg`:

```
xor <msg >newmsg
```

- Το νέο περιεχόμενο του `newmsg`:

```
rTSUR HIR NOK QORN _IST UCETCRU, QNI, QNCH JC@R  
GJIHC OH _IST TIIK, RSTHU IPCT _IST VGVCTU.
```

--LINGHH mGUVGT jGPGRCT (1741-1801)



Παράδειγμα (Κρυπτογράφηση XOR)

- Το πρόγραμμα `xor.c` που ακολουθεί δεν αλλάζει όλους τους χαρακτήρες.
- Αν χρησιμοποιήσουμε το XOR σε αυτούς τους χαρακτήρες με κλειδί το `&` θα παρήγαγε αόρατους χαρακτήρες ελέγχου, οι οποίοι θα μπορούσαν να προκαλέσουν προβλήματα σε ορισμένα λειτουργικά συστήματα.
- Οπότε το πρόγραμμα ελέγχει αν ο αρχικός χαρακτήρας και ο νέος (κρυπτογραφημένος) χαρακτήρας εκτυπώνονται με χαρακτήρες.
- Εάν όχι, το πρόγραμμα θα γράψει τον αρχικό χαρακτήρα αντί για το νέο χαρακτήρα.



Παράδειγμα (Κρυπτογράφηση XOR)

```
/* Performs XOR encryption */  
  
#include <ctype.h>  
#include <stdio.h>  
  
#define KEY '&'  
  
int main(void)  
{  
    int orig_char, new_char;  
  
    while ((orig_char = getchar()) != EOF) {  
        new_char = orig_char ^ KEY;  
        if (isprint(orig_char) && isprint(new_char))  
            putchar(new_char);  
        else  
            putchar(orig_char);  
    }  
  
    return 0;  
}
```



Παράδειγμα 2 (Κρυπτογράφηση XOR)

```
#include <stdio.h>
#include <stdlib.h>
#define KEY '&'
int main(int argc, char *argv[])
{
    FILE *input_fp, *output_fp;
    int ch;

    if (argc != 3) {
        fprintf(stderr, "usage: xor input output\n");
        exit(EXIT_FAILURE);
    }
    if ((input_fp = fopen(argv[1], "rb")) == NULL) {
        fprintf(stderr, "Can't open %s\n", argv[1]);
        exit(EXIT_FAILURE);
    }
    // Continued..
```

"r", and "w" to portably read/write text files on any system. That guarantees that the files are written and read properly.

"rb" and "wb" to read/write a binary file, so that an unfortunate newline-translation doesn't mess your data.



Παράδειγμα 2 (Κρυπτογράφηση XOR)

```
// ...  
  
if ((output_fp = fopen(argv[2], "wb")) == NULL) {  
    fprintf(stderr, "Can't open %s\n", argv[2]);  
    fclose(input_fp);  
    exit(EXIT_FAILURE);  
}  
  
while ((ch = getc(input_fp)) != EOF)  
    putc(ch ^ KEY, output_fp);  
  
fclose(input_fp);  
fclose(output_fp);  
return 0;  
}
```

putc - writes a character (an unsigned char) specified by the argument char to the specified stream and advances the position indicator for the stream.

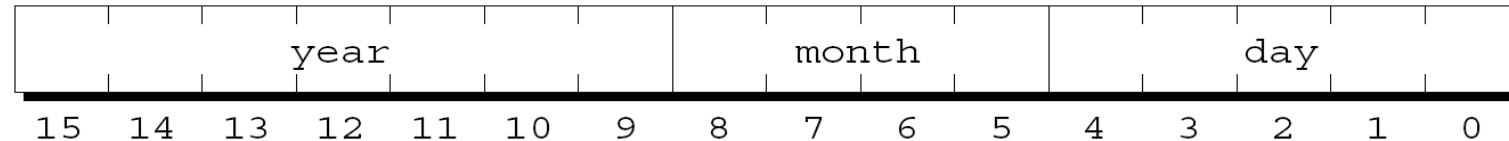


Δυαδικά Πεδία σε Δομές

- Μέχρι στιγμής υποθέσαμε ότι όλες οι **προσβάσεις σε bits** γίνονταν σε κάποιο **μη-προσημασμένο ακέραιο** (unsigned int).
- Κάτι τέτοιο μπορεί να δυσχεραίνει την **αναγνωσιμότητα** του προγράμματος μας ακόμη και να κάναμε τα ακόλουθα:
 - **Τεκμηρίωση** δυαδικής λογικής με σχόλια.
 - **Ονομασία των bit flags** (π.χ., BLUE, RED, ...)
- Τώρα θα δούμε ένα πρακτικό μηχανισμό για **διαχείριση bits** σε **προγράμματα**.
 - Ειδικότερα θα δηλώσουμε δομές των οποίων τα πεδία θα είναι **bit ακολουθίες** αντί **byte** ακολουθίες.

Δυαδικά Πεδία σε Δομές

- **Παράδειγμα:** Πως αποθηκεύει το DOS την ημερομηνία κατά την οποία δημιουργήθηκε/ άλλαξε για τελευταία φορά ένα αρχείο.
 - Εφόσον οι **ημέρες, μήνες** και τα **έτη** λαμβάνουν μικρές τιμές, αποθηκεύοντας τα ως ακέραιους θα **σπαταλούσε** πολύ χώρο (σε μια εποχή που αυτός ο χώρος δεν υπήρχε 1980: MSDOS / FAT12 / 8MB!!!) 😞.
 - Αντί αυτού, το DOS δεσμεύει μόνο 16 bits για μια ημερομηνία, **5 bits** για την μέρα (μέχρι **32 μέρες**), **4 bits** για **τον μήνα** (μέχρι **16 μήνες**), και 7 bits για τον χρόνο (μέχρι 128 χρόνια):



Δυαδικά Πεδία σε Δομές

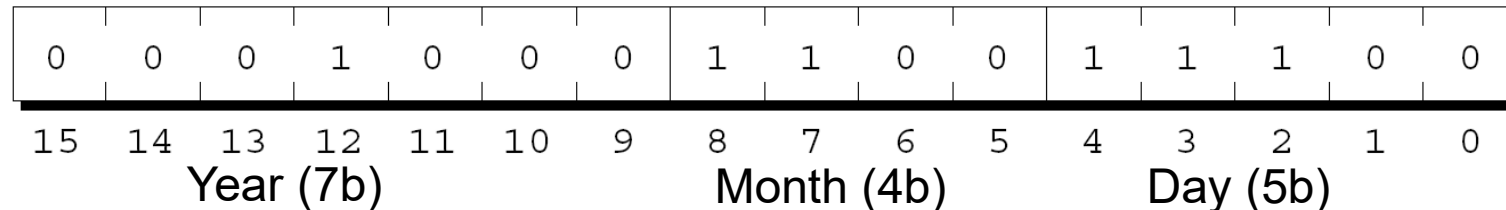
- Δήλωση μελών μιας δομής:

```
typedef struct {  
    unsigned int day: 5;    // 5 bits  
    unsigned int month: 4; // 4 bits  
    unsigned int year: 7;  // 7 bits  
} FILE_DATE;
```

- Χρήση μέλους μιας δομής:

```
struct file_date fd;  
fd.day = 28;  
fd.month = 12;  
fd.year = 8; /* 1988 (μέγιστο: 1980 + 127 χρόνια = 2107) */
```

- Αναπαράσταση της δομής `fd` μετά την ανάθεση:



Δυαδικά Πεδία σε Δομές

```
#include <stdio.h> // printf
#define SYEAR 1980
typedef struct {
    unsigned int day: 5;    // 5 bits
    unsigned int month: 4; // 4 bits
    unsigned int year: 7;   // 7 bits
} FILE_DATE;
```

```
int main(void) {
    FILE_DATE fd;
    // α) 1988 (max year: 1980+127 = 2107)
    fd.day = 28;    fd.month = 12;    fd.year = 8;
    printf("Date: %d/%d/%d\n", (int) fd.day, (int) fd.month,
    ((int) fd.year)+SYEAR);
    printf("Size: %ld\n", sizeof(FILE_DATE));
    return 0;
}
```

α) Δήλωση bit πεδίων

Επιστρέφει

Date: 28/12/1988

Size: 4

ε) Γιατί όχι SIZE=2; Θα εξηγηθεί σε λίγο

β) Τέχνασμα για μείωση χώρου!

γ) Μετατροπή Τύπου για εκτύπωση

δ) Απαραίτητο για LP64 μοντέλο



Δυαδικά Πεδία σε Δομές

- Σημειώστε ότι ο γνωστός **τελεστής διεύθυνσης (&)** ΔΕΝ μπορεί να εφαρμοστεί πάνω σε ένα πεδίο bit.

- Αυτό, εφόσον τα δεδομένα στην μνήμη έχουν **διευθύνσεις σε bytes** για λόγους επίδοσης

- εναλλακτικά ο pointer θα έπρεπε να είναι ακόμη μεγαλύτερος από 4 και 8 bytes!

- Συνεπώς, συναρτήσεις όπως το `scanf` δεν μπορούν να αποθηκεύουν **απευθείας** ένα πεδίο bit:

```
scanf ("%d", &fd.day);          /*** ΛΑΘΟΣ ***/
```

- Αλλά, μπορούμε να το κάνουμε έμμεσα...

```
int day;                        /*** ΟΡΘΟ ***/
```

```
scanf ("%d", &day);
```

```
fd.day = (unsigned int) day;
```



Δυαδικά Πεδία σε Δομές

- Η αναφορά σε bits γίνεται πλέον με **εύληπτο τρόπο**.
 - `fd.day` παρά `(data & 0xF800) // 0xF800=11111000 00000000`
- Ο τύπος ενός bit-field πρέπει να είναι τύπου:
unsigned int (π.χ., `short/long/long long` κτλ.)
- Στην C99, bit-fields μπορεί επίσης να έχουν τύπο `_Bool` (το οποίο είναι ουσιαστικά και πάλι ένας `unsigned int`)
- Σημειώστε ότι η δομή καταλαμβάνει, “κανονικά”, **16 bits** δηλαδή **2 bytes**, ενώ στο παράδειγμα επέστρεψε η `sizeof(fd)` μέγεθος 4 bytes.
 - Αυτό οφείλεται σε λόγους **ευθυγράμμισης δεδομένων στη μνήμη (memory alignment)**, το οποίο είχαμε δει και όταν συζητήσαμε τα struct της C.



Δυαδικά Πεδία σε Δομές

- Ερχόμενοι πίσω στο παράδειγμα των bit δομών, παρατηρούμε ότι **έχουμε** το θέμα της **ευθυγράμμισης** το οποίο είχαμε δει στη **διάλεξη 8 (structs)**, δηλ.,:

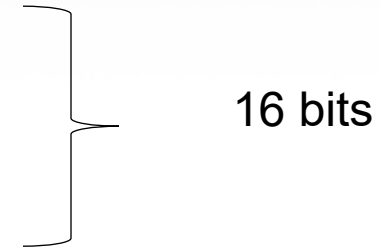
```
typedef struct {
    unsigned int day: 5;    // 5 bits
    unsigned int month: 4; // 4 bits
    unsigned int year: 7;  // 7 bits
} FILE_DATE;
```

- Συνεπώς, `sizeof(FILE_DATE)` επιστρέφει **4 byte** (θεωρώντας ότι η μνήμη έχει 4-byte ευθυγράμμιση): δηλ., 16 bit ωφέλιμα και 16 bit padding)
- Τι θα γίνει εάν προσθέσουμε το **`__attribute__((__packed__))`** όρισμα;



Δυαδικά Πεδία σε Δομές και Ευθυγράμμιση Μνήμης

```
typedef struct {  
    unsigned int day: 5;    // 5 bits  
    unsigned int month: 4; // 4 bits  
    unsigned int year: 7;  // 7 bits  
} __attribute__((__packed__)) FILE_DATE;
```



- Τώρα, `sizeof(FILE_DATE)` επιστρέφει **2 Bytes**, επειδή η μνήμη στο παράδειγμα μας δεν χρησιμοποιεί padding (δηλ., 16 bit ωφέλιμα και 0 bit padding)
- Τι θα γινόταν εάν κάποιο από τα πεδία ήταν 1 bit μεγαλύτερο (δηλ., το struct να χρειαζόταν 17 bits);
 - Τότε θα `sizeof(FILE_DATE)` επιστρέφει **3 byte**.
 - Μην ξεχνάτε ότι η μνήμη έχει διευθύνσεις σε bytes και όχι bits!



Δυαδικά Πεδία σε Δομές και Ευθυγράμμιση Μνήμης

- Ο μεταγλωττιστής λοιπόν, **πακετάρει (packs)** τα πεδία bits ένα-ένα σε μια μονάδα αποθήκευσης (storage unit), **χωρίς κενά μεταξύ των πεδίων**
 - π.χ., packing `fd.day` (5 bit) σε `storage_unit` (16 bit), δηλ., 0000 0000 0001 1111 ή 1111 1000 0000 0000
 - Η **διάταξη** των bits μέσα στο **storage_unit** (αριστερά-προς-δεξιά ή δεξιά-προς-αριστερά) ορίζεται από την υλοποίηση (**implementation-defined**) αλλά δεν μας απασχολεί προς το παρόν (υποθέστε ότι είναι από αριστερά)
- Για την αναπαράσταση του επόμενου πεδίου οι μεταγλωττιστές κάνουν ένα από τα ακόλουθα:
 - A. Μετακινούνται** στην **αρχή** του επόμενου storage unit, ή
 - B. Επεκτείνουν** το πεδίο bit **μεταξύ** των storage units εάν απαιτείται



Αποθήκευση Δομών από Bits (Συνοδευτικά Ορίσματα)

- Το όνομα ενός bit-field μπορεί να παραλειφθεί.
 - Π.χ., μια δομή όπου παραλείπεται το seconds:

```
struct file_time {  
    unsigned int : 5; /* not used */  
    unsigned int minutes: 6;  
    unsigned int hours: 5;  
};
```

- Κάτι τέτοιο συνεχίζει να **δεσμεύει τον εν λόγω χώρο στη δομή (απλά δεν έχει όνομα!)**
- Αυτό μπορεί να χρησιμοποιηθεί σε περιπτώσεις που **είναι undefined κάποια πεδία** αλλά υπάρχουν (π.χ., σε ένα file format που δεν θέλουμε να επεξεργαστούμε τα νωρίτερα bytes).
- Τα ανώνυμα δυαδικά πεδία χρησιμοποιούνται ως «padding» για να εξασφαλιστεί ότι τα υπόλοιπα πεδία είναι σωστά τοποθετημένα.



Αποθήκευση Δομών από Bits (Συνοδευτικά Ορίσματα)

- Το μήκος ενός μη-δηλωμένου bit-field μπορεί να είναι 0:

```
typedef struct {  
    unsigned int a: 4;  
    unsigned int : 0; /* 0-length bit-field */  
    unsigned int b: 8;  
} TEST;
```

- Ένα **μήκος-0** bit-field δηλώνει του μεταγλωττιστή να **ευθυγραμμίσει ρητά** το επόμενο bit-field στην αρχή ενός νέου block μνήμης.

- Δηλαδή, **sizeof (TEST)** επιστρέφει **8 Bytes** (θεωρώντας ότι η μνήμη έχει 4-byte ευθυγράμμιση)

- δηλ., $4 \text{ bit (ωφέλιμα)} + 28 \text{ bit (padding)} + 8 \text{ bit (ωφέλιμα)} + 24 \text{ bit (padding)}$

4 Bytes

4 Bytes

Αποθήκευση Δομών από Bits (Συνοδευτικά Ορίσματα)

- Μη-δηλωμένο μηδενικό bit-field με attr..packed:

```
typedef struct {  
    unsigned int a: 4;  
    unsigned int : 0;    /* 0-length bit-field */  
    unsigned int b: 8;  
} __attribute__((__packed__)) TEST;
```

- Τώρα, **sizeof (TEST)** επιστρέφει **5 bytes** (θεωρώντας ότι η μνήμη έχει 4-byte ευθυγράμμιση)

- δηλ., 4 bit (ωφέλιμα) + 28 bit (padding) + 8 bit (ωφέλιμα)+NO extra padding!

4 Bytes

1 Byte

