

Αρχές Προγραμματισμού

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

Η δομή

```
struct <όνομα δομής> {  
    <τύπος 1ου μέλους> <όνομα 1ου μέλους>;  
    <τύπος 2ου μέλους> <όνομα 2ου μέλους>;  
    <τύπος 3ου μέλους> <όνομα 3ου μέλους>;  
    ...  
    <τύπος ηου μέλους> <όνομα ηου μέλους>;  
} <λίστα ονομάτων μεταβλητών> ;
```

Ορισμός τύπου **struct** address

```
struct address {  
    char street[20];  
    int number;  
    int code;  
    char city[20];  
};
```

Παράδειγμα

```
struct address {  
    char street[20];  
    int number;  
    int code;  
    char city[20];  
} myaddress;
```

συνδυασμένος
ορισμός τύπου
struct address
και δήλωση
μεταβλητής
τύπου **struct**
address

```
struct address myaddress ;
```

όνομα μεταβλητής τύπου
struct address

χρήση τύπου **struct** address

Αρχικοποίηση μεταβλητών

```
struct address {  
    char street[20];  
    int number;  
    int code;  
    char city[20];  
} ;
```

```
struct address myaddress = {"Anthewn", 1, 123,  
    "Patra" };
```

typedef

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

```
struct address {
    char street[20];
    int number;
    int code;
    char city[20];
};
typedef struct address Address;
```

```
void report (Address) ;
Address readaddress (void) ;
```

```
main ( ) {
```

```
    Address myaddress ;
```

```
        myaddress = readaddress( );
        report (myaddress);
```

```
}
```

```
Address readaddress (void) {
    Address localaddress;
```

```
    printf("Odos:\t");
    scanf("%s", localaddress.street);
    printf("Ar. :\t");
    scanf("%d", &localaddress.number);
    printf("Code:\t");
    scanf("%d", &localaddress.code);
    printf("Poli:\t");
    scanf("%s", localaddress.city);
```

```
    return localaddress;
```

```
}
```

```
void report (Address local) {
    printf("Odos: %20s\n", local.street);
    printf("Ar. : %20d\n", local.number);
    printf("T.C.: %20d\n", local.code);
    printf("Poli: %20s\n", local.city);
```

```
}
```

Τι είναι ταχύτερο;

```
#include <stdio.h>
#include <time.h>
#define TIMES 1000000

typedef struct test {
    char data[100];
} Test;

void byvalue(Test a) {
    Test b;
    /* do something */
}

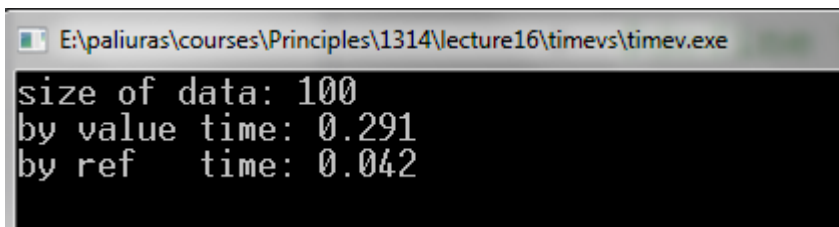
void byref(Test *a ) {
    Test b;
    /* do something */
}
```

```
int main( void) {
    Test a ;
    int i;
    clock_t start, stop;

    start = clock();
    printf("size of data: %d\n", sizeof (Test));
    for (i=0;i<TIMES; i++)
        byvalue(a);
    stop = clock();
    printf("by value time: %g\n",
        (double) (stop - start)/CLOCKS_PER_SEC);

    start = clock();
    for (i=0;i<TIMES; i++)
        byref(&a);
    stop = clock();
    printf("by ref  time: %g\n",
        (double) (stop - start)/CLOCKS_PER_SEC);

    return 0;
}
```



```
E:\paliuras\courses\Principles\1314\lecture16\timevs\timev.exe
size of data: 100
by value time: 0.291
by ref  time: 0.042
```

Κλήση με αξία και
Κλήση με αναφορά

Ένθεση Δομών

```
struct person {  
    char firstname[20];  
    char surname[20];  
    int age;  
    struct address homeaddress;  
    struct address bussinessaddress;  
} ;
```

Δείκτες ως μέλη δομών

```
struct person {  
    char firstname[20];  
    char surname[20];  
    int age;  
    struct address homeaddress;  
    struct address bussinessaddress;  
    struct person *next_person;  
} ;
```

Δείκτης σε δομή τύπου person

ΣΥΝΟΠΤΙΚΑ ΟΙ ΔΟΜΕΣ (1)

- Ορισμός δομής \Rightarrow ορισμός τύπου

```
struct test {  
    int a;  
    char d[10]; };  
struct test mytest;
```

- Επιστρέφονται και περνούν κατ' αξία από συναρτήσεις.

```
struct test dosomething(int a, struct test b) {  
    <κώδικας>  
}
```

- Βοηθάει το **typedef**

```
typedef struct test Test;  
Test dosomething(int a, Test b) {  
    <κώδικας>  
}
```

Συνοπτικά οι δομές (2)

- Αναφερόμαστε με dot notation σε μέλος μιας μεταβλητής τύπου δομής

```
struct test {  
    int a;  
    char d[10]; };
```

ΤΥΠΟΣ

Test mytest;

mytest.a = 5;

strcpy(mytest.d, "hello");

μεταβλητή

μέλος

- Ένα μέλος μιας μεταβλητής τύπου μίας δομής έχει ως τύπο τον τύπο του μέλους.

```

#include <stdio.h>
#include <stdlib.h>
struct data {
    int * data_ptr ;
    int len;
};
typedef struct data Data;
Data createdata(void);
void printdata(Data);

int main ( void ) {

    Data mydata;

    mydata = createdata();
    printdata(mydata);
    /* how to free ?*/
    return 0;
}

```

```

Data createdata (void) {
    Data data ;
    int x;
    int i;
    scanf("%d", &x);
    data.data_ptr = malloc(x*sizeof (int));
    data.len = x;
    for (i=0;i<x;i++) data.data_ptr[i] = 2*i;
    return data;
}

void printdata(Data data) {
    int i;
    for (i=0;i<data.len;i++) printf("%d\n", data.data_ptr[i]);
    return ;
}

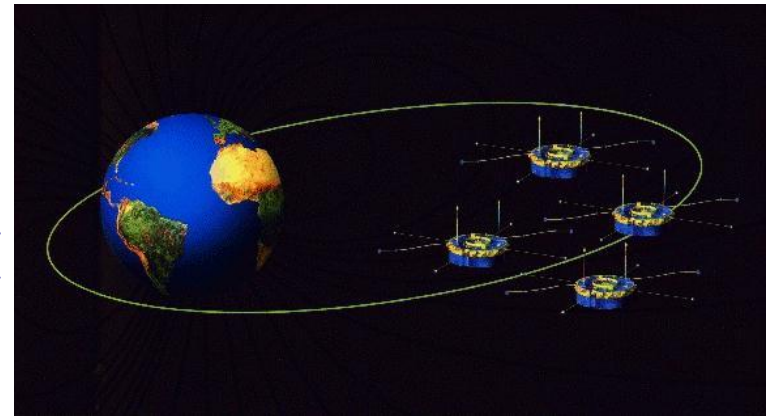
```

Σύστημα τύπων:

Ένα βήμα στην ανάπτυξη λογισμικού

- Καθορισμός συστήματος τύπων οι οποίοι χαρακτηρίζουν τα δεδομένα του προβλήματος
 - Επιβάλλει κανόνες στη χρήση των δεδομένων
 - Αυτοματοποιεί τον λογικό έλεγχο
- Για να αξιοποιήσουμε αυτή τη δυνατότητα
 - ορίζουμε κατάλληλους τύπους
 - γράφουμε κατάλληλο κώδικα

4 Ιουνίου 1996, Κουρού, Γαλλ. Γουιιάνα



Η πρώτη πτήση του Ariane 5G με φορτίο τέσσερεις δορυφόρους Cluster.

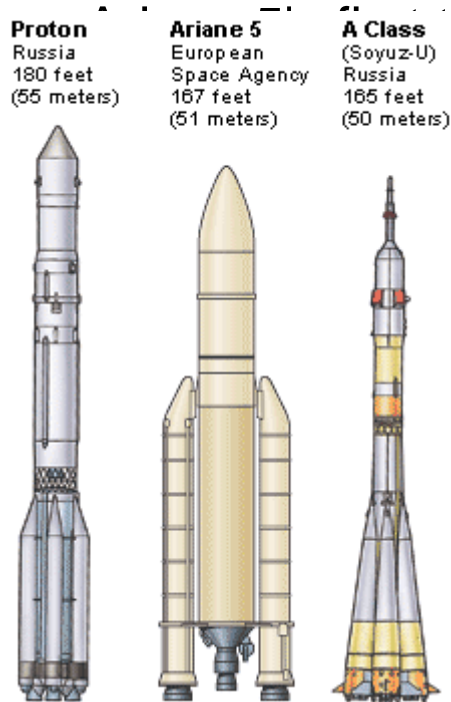


Ariane takes off Thursday
(Ariane TV)



Smoke from the explosion
June 4, 1996 (AP Photo)

Ariane 5 Flight 501



First flight ([Ariane 5 Flight 501](#)) on [4 June 1996](#)
rocket self-destructing 37 seconds after launch
malfunction in the control software, which was
the most expensive [computer bugs](#) in history.

Conversion from 64-bit [floating point](#) to 16-bit [signed](#)
integer caused a processor trap (operand error).

That number had a value too large to be
a 16-bit signed integer.

Conversions had led to the disabling of the
software handler (in [Ada](#) code) for this trap, although other
conversions of comparable variables in the code remained
protected.

Κώδικας σε ADA

```
...  
declare  
vertical_veloc_sensor: float;  
horizontal_veloc_sensor: float;  
vertical_veloc_bias: integer;  
horizontal_veloc_bias: integer;  
...  
begin  
declare  
pragma suppress(numeric_error, horizontal_veloc_bias);  
  begin  
    sensor_get(vertical_veloc_sensor);  
    sensor_get(horizontal_veloc_sensor);  
    vertical_veloc_bias := integer(vertical_veloc_sensor);  
    horizontal_veloc_bias := integer(horizontal_veloc_sensor);  
  ...  
  exception  
    when numeric_error => calculate_vertical_veloc();  
    when others => use_irs1();  
  end;  
end irs2;
```

Date & Time (UTC)	Flight	Model	Serial number	Payload	Result
04.06.1996 12:34:06	V-89	Ariane-5G	501	Cluster	Failure
30.10.1997 13:43:00	V-101	Ariane-5G	502	MaqSat H & TEAMSAT, MaqSat B, YES	Partial failure
21.10.1998 16:37:21	V-112	Ariane-5G	503	MaqSat 3, ARD	Success
10.12.1999 14:32:07	V-119	Ariane-5G	504	XMM-Newton	Success
21.03.2000 23:28:19	V-128	Ariane-5G	505	INSAT 3B , AsiaStar	Success
14.09.2000 22:54:07	V-130	Ariane-5G	506	Astra 2B, GE 7	Success
16.11.2000 01:07:07	V-135	Ariane-5G	507	PAS 1R, Amsat P3D , STRV 1C, STRV 1D	Success
20.12.2000 00:26:00	V-138	Ariane-5G	508	Astra 2D , GE 8 (Aurora 3), LDREX	Success
08.03.2001 22:51:00	V-140	Ariane-5G	509	Eurobird 1 , BSat 2a	Success
12.07.2001 22:58:00	V-142	Ariane-5G	510	Artemis, BSat 2b	Partial failure
01.03.2002 01:07:59	V-145	Ariane-5G	511	Envisat	Success
05.07.2002 23:22:00	V-153	Ariane-5G	512	Stellat 5, N-Star c	Success
28.08.2002 22:45:00	V-155	Ariane-5G	513	Atlantic Bird 1, MSG 1, MFD	Success
11.12.2002 22:22:00	V-157	Ariane-5ECA	517	Hot Bird 7, Stentor, MFD A, MFD B	Failure
09.04.2003 22:52:19	V-160	Ariane-5G	514	Insat 3A, Galaxy 12	Success
11.06.2003 22:38:15	V-161	Ariane-5G	515	Optus C1 , BSat 2c	Success
27.09.2003 23:14:46	V-162	Ariane-5G	516	Insat 3E, eBird 1, SMART-1	Success
02.03.2004 07:17:44	V-158	Ariane-5G+	518	Rosetta	Success
18.07.2004 00:44:00	V-163	Ariane-5G+	519	Anik F2	Success
18.12.2004 16:26:00	V-165	Ariane-5G+	520	Helios 2A, Essaim 1, 2, 3 and 4, PARASOL , Nanosat 01	Success
12.02.2005 21:03:00	V-164	Ariane-5ECA	521	XTAR-EUR, Maqsat B2, Sloshsat	Success
11.08.2005 08:20:00	V-166	Ariane-5GS	523	Thaicom 4-iPStar 1	Success
13.10.2005 22:32:00	V-168	Ariane-5GS	524	Syracuse 3A, Galaxy 15	Success
16.11.2005 23:46:00	V-167	Ariane-5ECA	522	Spaceway F2 , Telkom 2	Success
21.12.2005 22:33:00	V-169	Ariane-5GS	525	Insat 4A , MSG 2 , MFD C	Success
11.03.2006 22:32:50	V-170	Ariane-5ECA	527	Spainsat, MFD C, MFD C, Hot Bird 7A	Success
26.05.2006 21:08:50	V-171	Ariane-5ECA	529	Satmex 6, Thaicom 5	Success
11.08.2006 22:15:00	V-172	Ariane-5ECA	531	JCSat 10, Syracuse 3B	Success
13.10.2006 20:56:00	V-173	Ariane-5ECA	533	DirecTV-9S, Optus D1 , LDREX-2	Success
08.12.2006 22:08:00	V-174	Ariane-5ECA	534	WildBlue 1 , AMC 18	Success
11.03.2007 22:03	V-175	Ariane-5ECA	535	Skynet-5A, Insat-4B	Success
04.05.2007 22:29	V-176	Ariane-5ECA	536	Astra 1L, Galaxy 17	Success

Τύποι Δεδομένων

- Καθορίζουν σύνολο ιδιοτήτων μεταβλητών
 - πχ. το άθροισμα το ακεραίων είναι ακέραιος
- Αξιοποιούνται από ένα μηχανισμό ελέγχου
 - Τι γίνεται αν συνδυάσω σε μια έκφραση δεδομένα διαφορετικού τύπου;
 - Είναι σωστές οι διεπαφές των δομικών στοιχείων;
 - Βοηθάει στην αποφυγή λογικών λαθών στον κώδικα.

Έλεγχος Τύπων

Type Checking

- Κατά τη μεταγλώττιση παράγονται μηνύματα λάθους/προειδοποίησης όταν υπάρχει ασυμφωνία μεταξύ κλήσης και υλοποίησης συνάρτησης
 - μεταξύ του πλήθους των ορισμάτων,
 - των τύπων των ορισμάτων,
 - του επιστρεφόμενου τύπου.
- Βασική τεχνική αποφυγής σφαλμάτων
 - ενεργοποιείται με τη δήλωση προτύπων των συναρτήσεων
 - στην αρχή ή περιλαμβάνοντας κατάλληλα αρχεία κεφαλίδας (header files, .h)
 - Στη C μπορεί να προκαλέσει **έμμεσες** (implicit) μετατροπές τύπων.
- Γλώσσες με αυστηρό έλεγχο τύπων (strongly-typed languages)
 - Αποκλείουν τη χρήση δεδομένων τύπου T σε εκφράσεις οι οποίες δεν είναι κατασκευασμένες για να χειριστούν δεδομένα τύπου T.
- Γλώσσες με ασθενή έλεγχο τύπων
 - «Πρωτοβουλίες» σε μετατροπές τύπων.

Προστασία από λάθη λογικής μέσω ελέγχου τύπων

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

```
struct student {  
    char name[40];  
    int code;  
};
```

```
typedef struct student Student;
```

```
struct school {  
    char name[40];  
    int code;  
};
```

```
typedef struct school School;
```

```
main ( ) {  
    School myschool = {"paidikos",  
        10};  
    Student astudent =  
        {"giannakis", 4},  
        students[10];
```

```
students[0] = astudent;  
students[1] = myschool;  
}
```

error: incompatible types in assignment

λάθος: ασύμβατοι τύποι σε ανάθεση

Ο έλεγχος γίνεται την ώρα της μεταγλώττισης
(compile-time).

Όμως...

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

typedef int Oranges;
typedef int Apples;

int main ( void) {
    Oranges a, b;
    Apples c, d;
    int e = 0;

    a = b = 1;
    c = d = 1;
    d = a;
    b = d + e;
    printf("%d %d\n", d, b);
    return EXIT_SUCCESS;
}
```

- Καλή πρακτική για κώδικα που περιγράφει τη λειτουργία του
- Στη C, ο έλεγχος τύπων για βαθμωτούς τύπους δεν είναι αυστηρός.
 - Εδώ δεν παράγονται μηνύματα λάθους.
 - Γλώσσα weakly-typed

Έμμεσες αλλαγές τύπου στη C

```
short int s;          /* 2 bytes (στην υλοποίηση που χρησιμοποιούμε)*/
unsigned long int l; /* 4 bytes */
char c;              /* εξαρτάται από την υλοποίηση,
                    signed ή unsigned, 1 byte*/
float f;             /* συνήθως απλής ακρίβειας κατά IEEE 754, 4
                    bytes*/
double d;           /* συνήθως διπλής ακρίβειας κατά IEEE 754, 8
                    bytes*/
...
s = l;              /* τα λιγότερο σημαντικά ψηφία του l ερμηνεύονται ως
                    προσημασμένος αριθμός*/
l = s;              /* γίνεται επέκταση προσήμου στον s μέχρι να γίνει ίσου
                    μήκους με τον l και στη συνέχεια ερμηνεύεται ως αριθμός
                    χωρίς πρόσημο */
s = c;              /* επέκταση προσήμου σε μήκος s */
f = l;              /* ο τύπος float είναι μεγαλύτερος από τον τύπο του l */
f = d;              /* εδώ μπορεί να έχουμε απώλεια ακρίβειας */
...
```

Με έλεγχο τύπων...

```
#include <stdio.h>
int report(int);
struct data {
    char name[10];
    int code;
};
```

```
typedef struct data Data;
```

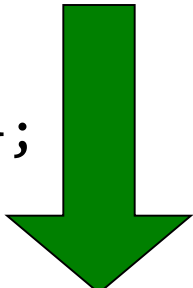
```
int main (void ) {
Data mydata = {"giannis", 2};
```

```
report(mydata);
return 0;
}
```

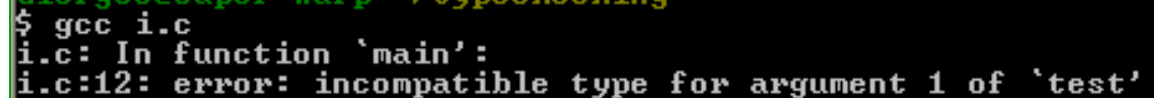
```
int report (int x){
    printf("%d\n",x);
return x;
}
```



ενεργοποίηση
μέσω δήλωσης



παράγεται μήνυμα λάθους
στη μεταγλώττιση
(compile-time error)



```
$ gcc i.c
i.c: In function 'main':
i.c:12: error: incompatible type for argument 1 of 'test'
```

Χωρίς έλεγχο τύπων

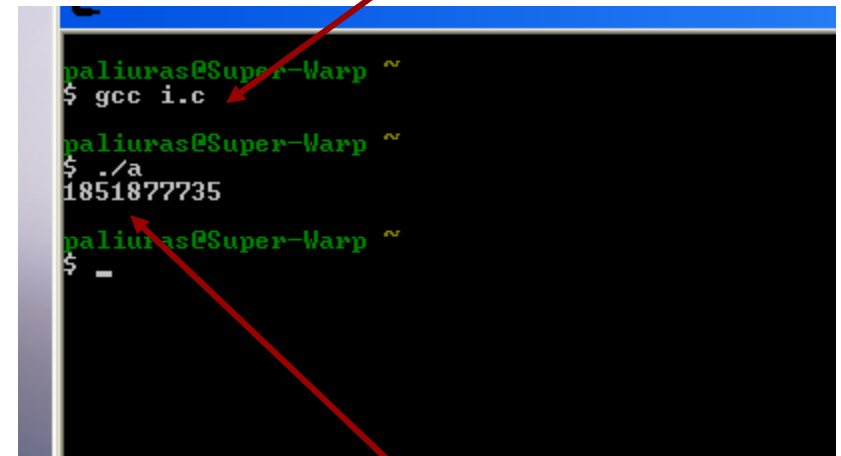
```
#include <stdio.h>
```

```
struct data {  
    char name[10];  
    int code;  
};  
typedef struct data Data;
```

```
main ( ) {  
    Data mydata = {"giannis", 2};  
  
    test(mydata);  
}
```

```
int test (int x){  
    printf("%d\n",x);  
    return x;  
}
```

όχι μήνυμα λάθους



A terminal window showing the compilation and execution of a C program. The user runs `gcc i.c` and then `./a`. The output is `1851877735`. The terminal prompt is `paliouras@Super-Warp ~`.

συμπεριφορά χωρίς νόημα, εδώ εύκολα εντοπίσιμη

Έλεγχος Τύπων – Παράδειγμα σε C


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

```
int main ( ) {  
    testfunc(1.1, 2.1) ;  
    testfunc(3);
```

```
    return 0;  
}
```

```
int testfunc (int a, int b) {  
    printf("%d %d\n", a, b);  
  
    return 0;  
}
```

έστω ότι **δεν**
δηλώνουμε την testfunc()



Ενεργοποίηση μηχανισμού

```
#include <stdio.h>
int testfunc(int, int );

int main ( void ) {
    testfunc(1.1, 2.1) ;
    testfunc(3);
    return 0;
}

int testfunc (int a, int b)
{

    printf("%d\n" ,a);

    return 0;
}
```

δήλωση συνάρτησης

ή ισοδύναμη ενέργεια

προκαλείται έμμεση αλλαγή τύπου
(implicit casting),
Μήνυμα λάθους για λάθος
πλήθος ορισμάτων

Θα πρέπει να χρησιμοποιούμε
κατάλληλα switches

Η σημασία των προειδοποιήσεων (warnings)

- gcc -Wall tc.c
 - Ή στα compiler options του Dev-C++
- Ο compiler προειδοποιεί για
 - υπονοούμενες δηλώσεις συναρτήσεων (implicit declarations).
 - Non-void functions χωρίς return ...
 - Και άλλα

Μετατροπές Τύπων: Παράδειγμα

```
#include <stdio.h>
int f(double);

int main (void ) {
    printf ("%g\n", (double) f(3) );
    return 0;
}

int f (double x) {
    double temp ;

    temp = x / 2;
    printf ("%g\n", temp);

    return temp ;
}
```

δύο κλήσεις της printf().
Τι τυπώνεται;

Προστασία από λάθη λογικής μέσω ελέγχου τύπων

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

```
struct student {  
    char name[40];  
    int code;  
};  
typedef struct student Student;
```

```
struct school {  
    char name[40];  
    int code;  
};  
typedef struct school School;
```

```
main ( ) {  
    School myschool = {"paidikos", 10};  
    Student astudent = {"giannakis", 4},  
        students[10];
```

```
    students[0] = astudent;
```

```
    students[1] = myschool;
```

```
}
```

error: incompatible types in assignment

λάθος: ασύμβατοι τύποι σε ανάθεση

Παράκαμψη (2)

```
#include <stdio.h>

struct student {
    char name[40];
    int code;
};
typedef struct student Student;

struct school {
    char name[40];
    int code;
};
typedef struct school School;

void report(Student);
```

```
main ( ) {
    School myschool = {"paidikos", 10};
    Student astudent = {"giannakis", 4},
        students[10];

    students[0] = astudent;
    students[1]= * ((Student *) &myschool);

    report(students[0]);
    report(students[1]);
}

void report(Student a) {
    printf("name: %s\n", a.name);
    printf("code: %4d\n", a.code);
}
```

Καλύτερα

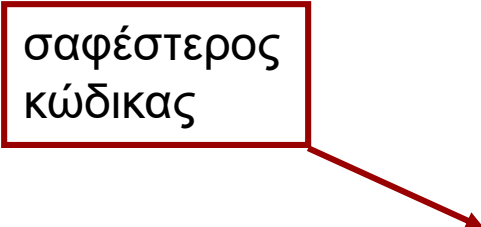
```
#include <stdio.h>
```

```
struct student {  
    char name[40];  
    int code;  
};  
typedef struct student Student;
```

```
struct school {  
    char name[40];  
    int code;  
};  
typedef struct school School;
```

```
void reportStudent(Student);  
void reportSchool(School);
```

σαφέστερος
κώδικας



```
main ( ) {  
    School myschool = {"paidikos", 10};  
    Student astudent = {"giannakis", 4},  
        students[10];
```

```
    students[0] = astudent;
```

```
    reportStudent(students[0]);  
    reportSchool(myschool);  
}
```

```
void reportStudent(Student a) {  
    printf("name: %s\n", a.name);  
    printf("code: %4d\n", a.code);  
}
```

```
void reportSchool(School a) {  
    reportStudent(*(Student *) &a);  
}
```

Ακόμα καλύτερα

```
#include <stdio.h>

struct student {
    char name[40];
    int code;
};
typedef struct student Student;

struct school {
    char name[40];
    int code;
};
typedef struct school School;

void reportStudent(Student);
void reportSchool(School);
Student ConvertSchooltoStudent(School);
```

```
main ( ) {
    School myschool = {"paidikos", 10};
    Student astudent = {"giannakis", 4},
        students[10];
    reportStudent(astudent);
    reportSchool(myschool);
}
```

```
void reportStudent(Student a) {
    printf("name: %s\n", a.name);
    printf("code: %4d\n", a.code);
}

void reportSchool(School a) {
    reportStudent(ConvertSchooltoStudent(a));
}

Student ConvertSchooltoStudent(School a) {
    return *((Student *) &a);
}
```

συνάρτηση για την αναλυτική αλλαγή τύπου.

- Ο έλεγχος τύπων προστατεύει από λογικά και άλλα λάθη
- Γράφουμε κώδικα που να αξιοποιεί τον έλεγχο τύπων
 - Η γλώσσα καθορίζει πώς να γράφουμε προγράμματα που να αξιοποιούν το σύστημα τύπων.