

Αντικειμενοστραφείς Γλώσσες Προγραμματισμού C++ / ROOT

Ιωάννης Παπαδόπουλος

Τμήμα Φυσικής, Πανεπιστήμιο Ιωαννίνων

Οκτώβριος 2018

- 1 Υπονοούμενες (implicit) μετατροπές Τύπων
- 2 Συναρτήσεις (Functions)
- 3 Δείκτες (Pointers)
- 4 Πίνακες (Arrays)

Υπονοούμενες (implicit) μετατροπές Τύπων

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 #include <iomanip> // για τη συνάρτηση setprecision
3 using namespace std;
4
5 int main(){
6     double d=-1.23456789012; // (1)
7     cout << setprecision(12); // (2)
8
9     cout << "d = " << d << endl; // (3)
10
11    float f = d;
12    cout << "f = " << f << endl; // (4)
13
14    int i = d;
15    cout << "i = " << i << endl; // (5)
16
17    unsigned short int j = i; // (6)
18    cout << "j = " << j << endl; // (7)
19
20    return 0;
21 }
```

- (1) Ορισμός του double d με πολλά ψηφία.
- (2) Το cout θα τυπώνει 12 σημαντικά ψηφία.
- (3) Οθόνη: d = -1.23456789012
- (4) Οθόνη: f = -1.23456788063
Η τιμή του d δεν μπορεί να αναπαρασταθεί ακριβώς από τον f, που έχει τύπο float και συνεπώς μικρότερη ακρίβεια.
- (5) Οθόνη: i = -1
Το δεκαδικό μέρος του αριθμού έχει απορριφθεί.
- (6) Ο ακέραιος j ορίζεται ως "unsigned short", και αρχικοποιείται στην τιμή του i.
- (7) Οθόνη: j = 65535 $\neq -1 \equiv i$
Η μετατροπή του i κατά την αρχικοποίηση του j (signed→unsigned) οδηγεί σε ανεπιθύμητο αποτέλεσμα...

- **Αναβάθμιση ακεραίου:**

Ένας char ή int (είτε εμπρόσημος είτε όχι) μπορεί να χρησιμοποιηθεί οπουδήποτε απαιτείται η χρήση ενός ακεραίου. Αν λοιπόν μία τιμή ενός αρχικού τύπου μπορεί να αναπαρασταθεί από έναν int (long int), τότε ο αρχικός τύπος μετατρέπεται σε int (long int). Σε αντίθετη περίπτωση, μετατρέπεται σε unsigned int (unsigned long int).

- **Μετατροπή ακεραίου:**

- Κατά τη μετατροπή **unsigned**→**signed**, αν η τιμή βρίσκεται εντός της περιοχής τιμών του τύπου signed, τότε αυτή διατηρείται. Διαφορετικά το αποτέλεσμα της μετατροπής είναι **απροσδιόριστο!**
- Στην περίπτωση της μετατροπής **signed**→**unsigned**, χρησιμοποιείται μία μέθοδος αποκοπής bits ή προσθήκης επιπλέον μηδενικών bits στις αναπαραστάσεις συμπληρώματος ως προς 2 (όταν ο τύπος unsigned έχει μεγαλύτερο εύρος). Μπορεί έτσι να οδηγήσει σε **ανεπιθύμητα αποτελέσματα**.

- **Μετατροπή δεκαδικού σε ακέραιο:**

Κατά τη μετατροπή αυτή, απορρίπτεται το δεκαδικό μέρος του αριθμού. Αν η τιμή που προκύπτει δεν μπορεί να αναπαρασταθεί από τον ακέραιο τύπο, τότε το αποτέλεσμα είναι **απροσδιόριστο!**

- **Μετατροπή ακεραίου σε δεκαδικό:**

Κατά τη μετατροπή αυτή, η τιμή του δεκαδικού αριθμού που προκύπτει μπορεί να είναι η αμέσως μεγαλύτερη ή μικρότερη τιμή που μπορεί να αναπαρασταθεί. Π.χ. ο αριθμός `unsigned int 429496729`, αν μετατραπεί σε `float` θα είναι ίσος με `429496736`, ενώ αν μετατραπεί σε `double` θα είναι ίσος με `429496729`.

- **Μετατροπές μεταξύ δεκαδικών:**

Όταν ένας λιγότερο ακριβής τύπος μετατρέπεται σε έναν ακριβέστερο τύπο (π.χ. `float`→`double`), η τιμή διατηρείται αμετάβλητη. Στην αντίθετη περίπτωση (π.χ. `double`→`float`), το αποτέλεσμα θα αντιστοιχηθεί στην αμέσως μεγαλύτερη ή μικρότερη τιμή που μπορεί να αναπαρασταθεί. Αν όμως η τιμή είναι εκτός των ορίων του λιγότερο ακριβούς τύπου, τότε το αποτέλεσμα είναι **απροσδιόριστο!**

Υπονοούμενες (implicit) μετατροπές Τύπων

- Αναβάθμιση τελεστών κατά την εκτέλεση αριθμητικών πράξεων:
 - Αν ο ένας τελεστής είναι long double, τότε ο άλλος τελεστής μετατρέπεται σε long double.
 - Διαφορετικά, αν ο ένας τελεστής είναι double, τότε ο άλλος τελεστής μετατρέπεται σε double.
 - Διαφορετικά, αν ο ένας τελεστής είναι float, τότε ο άλλος τελεστής μετατρέπεται σε float.
 - Διαφορετικά (όταν δηλαδή οι τελεστές είναι ακέραιοι), πραγματοποιείται αναβάθμιση ακεραίου και για τους δύο τελεστές, και έπειτα:
 - Αν ο ένας τελεστής είναι unsigned long int, τότε ο άλλος τελεστής μετατρέπεται σε unsigned long int.
 - Διαφορετικά, αν ο ένας τελεστής είναι long int και ο άλλος unsigned int, τότε το αποτέλεσμα εξαρτάται από το αν ο long int μπορεί να αναπαραστήσει όλες τις τιμές ενός unsigned int: Αν ναι, τότε ο τελεστής τύπου unsigned int μετατρέπεται σε long int. Αν όχι, μετατρέπονται και οι δύο τελεστές σε unsigned long int.
 - Διαφορετικά, αν ο ένας τελεστής είναι long int, τότε ο άλλος τελεστής μετατρέπεται σε long int.
 - Διαφορετικά, αν ο ένας τελεστής είναι unsigned int, τότε ο άλλος τελεστής μετατρέπεται σε unsigned int.
 - Διαφορετικά, και οι δύο τελεστές είναι τύπου int.

Υπονοούμενες (implicit) μετατροπές Τύπων

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4
5 int main(){
6     cout << setprecision(23);
7     for (int n : {12, 13, 14, 21, 23,
8                 26, 35, 171, 1754, 1755}) {
9         int          iP=1;
10        unsigned int  uiP=1;
11        long int      liP=1;
12        unsigned long int uliP=1;
13        float        fP=1;
14        double        dP=1;
15        long double   ldP=1;
16
17        for (          int c=1 ;(int)c<=n ;c++)  iP*=c;
18        for (    unsigned int c=1 ;(int)c<=n ;c++)  uiP*=c;
19        for (          long int c=1 ;(int)c<=n ;c++)  liP*=c;
20        for (unsigned long int c=1 ;(int)c<=n ;c++)  uliP*=c;
21        for (          float c=1 ;(int)c<=n ;c++)  fP*=c;
22        for (          double c=1 ;(int)c<=n ;c++)  dP*=c;
23        for (          long double c=1 ;(int)c<=n ;c++)  ldP*=c;
24
25        cout << endl;
26        cout << n << " ! =  iP = " <<  iP << endl;
27        cout << n << " ! =  uiP = " <<  uiP << endl;
28        cout << n << " ! =  liP = " <<  liP << endl;
29        cout << n << " ! =  uliP = " <<  uliP << endl;
30        cout << n << " ! =  fP = " <<  fP << endl;
31        cout << n << " ! =  dP = " <<  dP << endl;
32        cout << n << " ! =  ldP = " <<  ldP << endl;
33    }
34 }
```

Άσκηση:

Να υπολογιστεί το $n!$ για

$n \in \{12, 13, 14, 21, 23, 26, 35, 171, 1754, 1755\}$

χρησιμοποιώντας για το αποτέλεσμα αριθμούς

int,

unsigned int,

long int,

unsigned long int,

float,

double και

long double.

Συγκρίνετε τα αποτελέσματα.

Τι συμπέρασμα βγάζετε;

Δημιουργία του φακέλου ~/erg3 (μία φορά):

```
mkdir ~/erg3
```

Μετάβαση στον φάκελο που δημιουργήθηκε:

```
cd ~/erg3
```

Άνοιγμα και επεξεργασία αρχείου cpp με emacs:

```
emacs ask1.cpp &
```

Μεταγλώττιση (αφού σωθεί το αρχείο):

```
g++ -std=c++11 ask1.cpp -o ask1.exe
```

Εκτέλεση του προγράμματος που προκύπτει:

```
./ask1.exe
```

- Οι συναρτήσεις βελτιώνουν τον διαδικασιακό προγραμματισμό, συμβάλλουν στην άνετη ανάγνωση του κώδικα και στην ευκολότερη συντήρησή του.
- Μπορούν να ομαδοποιηθούν σε βιβλιοθήκες, που καλούνται από προγράμματα.

Συναρτήσεις

- Οι συναρτήσεις βελτιώνουν τον διαδικασιακό προγραμματισμό, συμβάλλουν στην άνετη ανάγνωση του κώδικα και στην ευκολότερη συντήρησή του.
- Μπορούν να ομαδοποιηθούν σε βιβλιοθήκες, που καλούνται από προγράμματα.

- Μία συνάρτηση πρέπει να δηλωθεί εντός του κώδικα πριν χρησιμοποιηθεί. Η δήλωση αυτή καθορίζει την υπογραφή της συνάρτησης.

```
1 // δήλωση της συνάρτησης
2 return_type myFunc(arg1_type, arg2_type, ... );
3
4 int main(){
5     ...
6     arg1_type x=... ;
7     arg2_type y=... ;
8     return_type z;
9     z=myFunc(x,y); // κλήση της συνάρτησης
10
11     return 0;
12 }
13
14 // ανάπτυξη της συνάρτησης
15 return_type myFunc(arg1_type A, arg2_type B, ... ) {
16     ...
17     return_type theResult=... ;
18     ...
19     // χρήση των A, B, ... σε εκφράσεις
20     ...
21     return theResult;
22 }
```

Συναρτήσεις

- Οι συναρτήσεις βελτιώνουν τον διαδικασιακό προγραμματισμό, συμβάλλουν στην άνετη ανάγνωση του κώδικα και στην ευκολότερη συντήρησή του.
- Μπορούν να ομαδοποιηθούν σε βιβλιοθήκες, που καλούνται από προγράμματα.

```
1 // δήλωση της συνάρτησης
2 return_type myFunc(arg1_type, arg2_type, ... );
3
4 int main(){
5     ...
6     arg1_type x=... ;
7     arg2_type y=... ;
8     return_type z;
9     z=myFunc(x,y);    // κλήση της συνάρτησης
10
11     return 0;
12 }
13
14 // ανάπτυξη της συνάρτησης
15 return_type myFunc(arg1_type A, arg2_type B, ... ) {
16     ...
17     return_type theResult=... ;
18     ...
19     // χρήση των A, B, ... σε εκφράσεις
20     ...
21     return theResult;
22 }
```

- Η χρήση της γίνεται μέσω κλήσεων της (**function calls**) εντός άλλων συναρτήσεων, όπως π.χ. εντός της main.

Συναρτήσεις

- Οι συναρτήσεις βελτιώνουν τον διαδικασιακό προγραμματισμό, συμβάλλουν στην άνετη ανάγνωση του κώδικα και στην ευκολότερη συντήρησή του.
- Μπορούν να ομαδοποιηθούν σε βιβλιοθήκες, που καλούνται από προγράμματα.

```
1 // δήλωση της συνάρτησης
2 return_type myFunc(arg1_type, arg2_type, ... );
3
4 int main(){
5     ...
6     arg1_type x=... ;
7     arg2_type y=... ;
8     return_type z;
9     z=myFunc(x,y);    // κλήση της συνάρτησης
10
11     return 0;
12 }
13
14 // ανάπτυξη της συνάρτησης
15 return_type myFunc(arg1_type A, arg2_type B, ... ) {
16     ...
17     return_type theResult=... ;
18     ...
19     // χρήση των A, B, ... σε εκφράσεις
20     ...
21     return theResult;
22 }
```

- Η ανάπτυξη της υλοποιείται οπουδήποτε μετά τη δήλωσή της (όχι όμως εντός άλλης συνάρτησης). Πρέπει να έχει την **ίδια ακριβώς υπογραφή** με αυτή της δήλωσης (ίδιο τύπο επιστροφής, ίδιους τύπους ορισμάτων, ίδια διάταξή τους). Τα ορίσματα αποκτούν πλέον και **αναγνωριστικά**, που χρησιμοποιούνται σε εκφράσεις εντός της συνάρτησης.

Συναρτήσεις

- Οι συναρτήσεις βελτιώνουν τον διαδικασιακό προγραμματισμό, συμβάλλουν στην άνετη ανάγνωση του κώδικα και στην ευκολότερη συντήρησή του.
- Μπορούν να ομαδοποιηθούν σε βιβλιοθήκες, που καλούνται από προγράμματα.

```
1 // δήλωση της συνάρτησης
2 return_type myFunc(arg1_type, arg2_type, ... );
3
4 int main(){
5     ...
6     arg1_type x=... ;
7     arg2_type y=... ;
8     return_type z;
9     z=myFunc(x,y); // κλήση της συνάρτησης
10
11     return 0;
12 }
13
14 // ανάπτυξη της συνάρτησης
15 return_type myFunc(arg1_type A, arg2_type B, ... ) {
16     ...
17     return_type theResult=... ;
18     ...
19     // χρήση των A, B, ... σε εκφράσεις
20     ...
21     return theResult;
22 }
```

- Μία συνάρτηση πρέπει να δηλωθεί εντός του κώδικα πριν χρησιμοποιηθεί. Η δήλωση αυτή καθορίζει την **υπογραφή** της συνάρτησης.
- Η χρήση της γίνεται μέσω κλήσεων της (**function calls**) εντός άλλων συναρτήσεων, όπως π.χ. εντός της main.
- Η ανάπτυξή της υλοποιείται οπουδήποτε μετά τη δήλωσή της (όχι όμως εντός άλλης συνάρτησης). Πρέπει να έχει την **ίδια ακριβώς υπογραφή** με αυτή της δήλωσης (ίδιο τύπο επιστροφής, ίδιους τύπους ορισμάτων, ίδια διάταξή τους). Τα ορίσματα αποκτούν πλέον και **αναγνωριστικά**, που χρησιμοποιούνται σε εκφράσεις εντός της συνάρτησης.

- Τα **ορίσματα περνούν** στις συναρτήσεις ως **τιμές** (call by value), δηλαδή:
 - Δημιουργείται στη μνήμη ένα αντίγραφο για κάθε αρχικό (πραγματικό) όρισμα, όπως αυτό εμφανίζεται στην αντίστοιχη εντολής κλήσης της συνάρτησης.
 - Τα αντίγραφα αυτά χρησιμοποιούνται στις όποιες εκφράσεις εντός της συνάρτησης, και μπορούν να τροποποιηθούν εντός αυτής.
 - Τα αρχικά (πραγματικά) ορίσματα παραμένουν αμετάβλητα, ανεξαρτήτως του τι συμβαίνει στα αντίγραφα τους εντός της συνάρτησης.
- Υπάρχουν δύο περιπτώσεις (που θα μελετηθούν αργότερα) οι οποίες δίνουν τη δυνατότητα χειρισμού/μεταβολής των αρχικών (πραγματικών) ορισμάτων της συνάρτησης.
 - Κλήση συναρτήσεων χρησιμοποιώντας ορίσματα με δείκτες.
 - Κλήση συναρτήσεων χρησιμοποιώντας ορίσματα με αναφορές.

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 // δήλωση της συνάρτησης fmodulo
5 int fmodulo(int, int);
6 //-----
7 int main(){
8
9     int i=133;
10    int j=5;
11
12    cout << "i=" << i
13         << ", j=" << j << endl;
14
15    int k=fmodulo(i,3*j);
16
17    cout << "i=" << i
18         << ", j=" << j
19         << ", k=" << k <<endl;
20
21    return 0;
22 }
23 //-----
24 // ανάπτυξη της συνάρτησης fmodulo
25 int fmodulo(int i, int m) {
26
27     i=i%m;
28
29     return i;
30 }
31 }
32 //-----
```

5 Δήλωση της συνάρτησης fmodulo

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 // δήλωση της συνάρτησης fmodulo
5 int fmodulo(int, int);
6 //-----
7 int main(){
8
9     int i=133;
10    int j=5;
11
12    cout << "i=" << i
13         << ", j=" << j << endl;
14
15    int k=fmodulo(i,3*j);
16
17    cout << "i=" << i
18         << ", j=" << j
19         << ", k=" << k <<endl;
20
21    return 0;
22 }
23 //-----
24 // ανάπτυξη της συνάρτησης fmodulo
25 int fmodulo(int i, int m) {
26
27     i=i%m;
28
29     return i;
30
31 }
32 //-----
```

25-31 Ανάπτυξη της συνάρτησης fmodulo.
Τα αντίγραφα των ορισμάτων κλήσης
ονομάζονται εντός της ανάπτυξης i και j.

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 // δήλωση της συνάρτησης fmodulo
5 int fmodulo(int, int);
6 //-----
7 int main(){
8
9     int i=133;
10    int j=5;
11
12    cout << "i=" << i
13         << ", j=" << j << endl;
14
15    int k=fmodulo(i,3*j);
16
17    cout << "i=" << i
18         << ", j=" << j
19         << ", k=" << k <<endl;
20
21    return 0;
22 }
23 //-----
24 // ανάπτυξη της συνάρτησης fmodulo
25 int fmodulo(int i, int m) {
26
27     i=i%m;
28
29     return i;
30
31 }
32 //-----
```

27 Η τιμή του i μεταβάλλεται.

29 Η τιμή του i επιστρέφεται από τη συνάρτηση.

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 // δήλωση της συνάρτησης fmodulo
5 int fmodulo(int, int);
6 //-----
7 int main(){
8
9     int i=133;
10    int j=5;
11
12    cout << "i=" << i
13         << ", j=" << j << endl;
14
15    int k=fmodulo(i,3*j);
16
17    cout << "i=" << i
18         << ", j=" << j
19         << ", k=" << k <<endl;
20
21    return 0;
22 }
23 //-----
24 // ανάπτυξη της συνάρτησης fmodulo
25 int fmodulo(int i, int m) {
26
27     i=i%m;
28
29     return i;
30 }
31 }
32 //-----
```

- 15 Η συνάρτηση καλείται με ορίσματα i και $3*j$. Έτσι, η μεταβλητή αντίγραφο i της συνάρτησης παίρνει την τιμή του i της `main`, και η μεταβλητή αντίγραφο m της συνάρτησης παίρνει την τιμή $3*j$. Η συνάρτηση επιστρέφει την τιμή του αντιγράφου i , η οποία αποδίδεται στη μεταβλητή k .

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 // δήλωση της συνάρτησης fmodulo
5 int fmodulo(int, int);
6 //-----
7 int main(){
8
9     int i=133;
10    int j=5;
11
12    cout << "i=" << i
13         << ", j=" << j << endl;
14
15    int k=fmodulo(i,3*j);
16
17    cout << "i=" << i
18         << ", j=" << j
19         << ", k=" << k <<endl;
20
21    return 0;
22 }
23 //-----
24 // ανάπτυξη της συνάρτησης fmodulo
25 int fmodulo(int i, int m) {
26
27     i=i%m;
28
29     return i;
30 }
31 }
32 //-----
```

12-13 Οθόνη: i=133, j=5

17-19 Οθόνη: i=133, j=5, k=13

Υπερφόρτωση συναρτήσεων (**function overloading**):

- Οι συναρτήσεις επιτρέπεται να έχουν το ίδιο όνομα, αρκεί να έχουν διαφορετική λίστα ορισμάτων (διαφορετικούς τύπους ή αριθμό ορισμάτων).

Υπερφόρτωση συναρτήσεων (**function overloading**):

- Οι συναρτήσεις επιτρέπεται να έχουν το ίδιο όνομα, αρκεί να έχουν διαφορετική λίστα ορισμάτων (διαφορετικούς τύπους ή αριθμό ορισμάτων).
- Ο μεταγλωττιστής επιλέγει την κατάλληλη κάθε φορά συνάρτηση, ανάλογα με τον τρόπο κλήσης της (ορίσματα, τύποι ορισμάτων).

Υπερφόρτωση συναρτήσεων (**function overloading**):

- Οι συναρτήσεις επιτρέπεται να έχουν το ίδιο όνομα, αρκεί να έχουν διαφορετική λίστα ορισμάτων (διαφορετικούς τύπους ή αριθμό ορισμάτων).
- Ο μεταγλωττιστής επιλέγει την κατάλληλη κάθε φορά συνάρτηση, ανάλογα με τον τρόπο κλήσης της (ορίσματα, τύποι ορισμάτων).

```
double fcn () ;
```

- Ορίσματα: κανένα
Επιστρέφει: double

Υπερφόρτωση συναρτήσεων (**function overloading**):

- Οι συναρτήσεις επιτρέπεται να έχουν το ίδιο όνομα, αρκεί να έχουν διαφορετική λίστα ορισμάτων (διαφορετικούς τύπους ή αριθμό ορισμάτων).
- Ο μεταγλωττιστής επιλέγει την κατάλληλη κάθε φορά συνάρτηση, ανάλογα με τον τρόπο κλήσης της (ορίσματα, τύποι ορισμάτων).

```
double fcn ( ) ;
```

- Ορίσματα: κανένα
Επιστρέφει: double

```
double fcn ( int ) ;
```

- Ορίσματα: int
Επιστρέφει: double

Συναρτήσεις

Υπερφόρτωση συναρτήσεων (**function overloading**):

- Οι συναρτήσεις επιτρέπεται να έχουν το ίδιο όνομα, αρκεί να έχουν διαφορετική λίστα ορισμάτων (διαφορετικούς τύπους ή αριθμό ορισμάτων).
- Ο μεταγλωττιστής επιλέγει την κατάλληλη κάθε φορά συνάρτηση, ανάλογα με τον τρόπο κλήσης της (ορίσματα, τύποι ορισμάτων).

```
double fcn ( ) ;
```

- Ορίσματα: κανένα
Επιστρέφει: double

```
double fcn ( int ) ;
```

- Ορίσματα: int
Επιστρέφει: double

```
int fcn ( int, int ) ;
```

- Ορίσματα: int και int
Επιστρέφει: int

Συναρτήσεις

Υπερφόρτωση συναρτήσεων (**function overloading**):

- Οι συναρτήσεις επιτρέπεται να έχουν το ίδιο όνομα, αρκεί να έχουν διαφορετική λίστα ορισμάτων (διαφορετικούς τύπους ή αριθμό ορισμάτων).
- Ο μεταγλωττιστής επιλέγει την κατάλληλη κάθε φορά συνάρτηση, ανάλογα με τον τρόπο κλήσης της (ορίσματα, τύποι ορισμάτων).

```
double fcn ( ) ;
```

- Ορίσματα: κανένα
Επιστρέφει: double

```
double fcn ( int ) ;
```

- Ορίσματα: int
Επιστρέφει: double

```
int fcn ( int, int ) ;
```

- Ορίσματα: int και int
Επιστρέφει: int

```
double fcn ( int, double ) ;
```

- Ορίσματα: int και double
Επιστρέφει: double

Συναρτήσεις

Υπερφόρτωση συναρτήσεων (**function overloading**):

- Οι συναρτήσεις επιτρέπεται να έχουν το ίδιο όνομα, αρκεί να έχουν διαφορετική λίστα ορισμάτων (διαφορετικούς τύπους ή αριθμό ορισμάτων).
- Ο μεταγλωττιστής επιλέγει την κατάλληλη κάθε φορά συνάρτηση, ανάλογα με τον τρόπο κλήσης της (ορίσματα, τύποι ορισμάτων).

```
double fcn ( ) ;
```

- Ορίσματα: κανένα
Επιστρέφει: double

```
double fcn ( int ) ;
```

- Ορίσματα: **int**
Επιστρέφει: double

```
int fcn ( int, int ) ;
```

- Ορίσματα: int και int
Επιστρέφει: int

```
double fcn ( int, double ) ;
```

- Ορίσματα: int και double
Επιστρέφει: double

```
int fcn ( int ) ;
```

- Ορίσματα: **int**
Επιστρέφει: int

ΛΑΘΟΣ: ίδια υπογραφή με τη 2η περίπτωση, αν και με διαφορετικό επιστρεφόμενο τύπο.

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 int f (int, int ) ;
5 int f (int, int, int ) ;
6 double f (double, int ) ;
7 //int f (double, int ) ; // -> ΛΑΘΟΣ
8 //-----
9 int main(){
10 cout << "f(1,2) " << f(1,2) << endl;
11 cout << "f(1,2,3) " << f(1,2,3) << endl;
12 cout << "f(3.5,2) " << f(3.5,2) << endl;
13 cout << "f(1.9,2.9) " << f(1.9,2.9) << endl;
14 cout << "f(1,2.9) " << f(1,2.9) << endl;
15 cout << "f(1.9,2,3) " << f(1.9,2,3) << endl;
16 }
17 //-----
18 int f (int a,int b) {
19 cout << "-> A: i i => i : " ;
20 return a+b; }
21 //-----
22 int f (int a,int b,int c) {
23 cout << "-> B: i i i => i : " ;
24 return a+b+c; }
25 //-----
26 double f (double a,int b) {
27 cout << "-> C: d i => d : " ;
28 return a+b; }
```

10 f(1,2) -> A: i i => i : 3

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 int f (int, int ) ;
5 int f (int, int, int ) ;
6 double f (double, int ) ;
7 //int f (double, int ) ; // -> ΛΑΘΟΣ
8 //-----
9 int main(){
10 cout << "f(1,2) " << f(1,2) << endl;
11 cout << "f(1,2,3) " << f(1,2,3) << endl;
12 cout << "f(3.5,2) " << f(3.5,2) << endl;
13 cout << "f(1.9,2.9) " << f(1.9,2.9) << endl;
14 cout << "f(1,2.9) " << f(1,2.9) << endl;
15 cout << "f(1.9,2,3) " << f(1.9,2,3) << endl;
16 }
17 //-----
18 int f (int a,int b) {
19 cout << "-> A: i i => i : " ;
20 return a+b; }
21 //-----
22 int f (int a,int b,int c) {
23 cout << "-> B: i i i => i : " ;
24 return a+b+c; }
25 //-----
26 double f (double a,int b) {
27 cout << "-> C: d i => d : " ;
28 return a+b; }
```

12 f(1,2,3) -> B: i i i => i : 6

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 int f (int, int ) ;
5 int f (int, int, int ) ;
6 double f (double, int ) ;
7 //int f (double, int ) ; // -> ΛΑΘΟΣ
8 //-----
9 int main(){
10 cout << "f(1,2) " << f(1,2) << endl;
11 cout << "f(1,2,3) " << f(1,2,3) << endl;
12 cout << "f(3.5,2) " << f(3.5,2) << endl;
13
14 cout << "f(1.9,2.9) " << f(1.9,2.9) << endl;
15
16 cout << "f(1,2.9) " << f(1,2.9) << endl;
17
18 cout << "f(1.9,2,3) " << f(1.9,2,3) << endl;
19
20 }
21 //-----
22 int f (int a,int b) {
23 cout << "-> A: i i => i : " ;
24 return a+b; }
25 //-----
26 int f (int a,int b,int c) {
27 cout << "-> B: i i i => i : " ;
28 return a+b+c; }
29 //-----
30 double f (double a,int b) {
31 cout << "-> C: d i => d : " ;
32 return a+b; }
33
```

14 f(3.5,2) -> C: d i => d : 5.5

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 int f (int, int ) ;
5 int f (int, int, int ) ;
6 double f (double, int ) ;
7 //int f (double, int ) ; // -> ΛΑΘΟΣ
8 //-----
9 int main(){
10 cout << "f(1,2) " << f(1,2) << endl;
11 cout << "f(1,2,3) " << f(1,2,3) << endl;
12 cout << "f(3.5,2) " << f(3.5,2) << endl;
13 cout << "f(1.9,2.9) " << f(1.9,2.9) << endl;
14 cout << "f(1,2.9) " << f(1,2.9) << endl;
15 cout << "f(1.9,2,3) " << f(1.9,2,3) << endl;
16 }
17 //-----
18 int f (int a,int b) {
19 cout << "-> A: i i => i : " ;
20 return a+b; }
21 //-----
22 int f (int a,int b,int c) {
23 cout << "-> B: i i i => i : " ;
24 return a+b+c; }
25 //-----
26 double f (double a,int b) {
27 cout << "-> C: d i => d : " ;
28 return a+b; }
```

16 f(1.9,2.9) -> C: d i => d : 3.9

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 int f (int, int ) ;
5 int f (int, int, int ) ;
6 double f (double, int ) ;
7 //int f (double, int ) ; // -> ΛΑΘΟΣ
8 //-----
9 int main(){
10 cout << "f(1,2) " << f(1,2) << endl;
11 cout << "f(1,2,3) " << f(1,2,3) << endl;
12 cout << "f(3.5,2) " << f(3.5,2) << endl;
13 cout << "f(1.9,2.9) " << f(1.9,2.9) << endl;
14 cout << "f(1,2.9) " << f(1,2.9) << endl;
15 cout << "f(1.9,2,3) " << f(1.9,2,3) << endl;
16 }
17 //-----
18 int f (int a,int b) {
19 cout << "-> A: i i => i : " ;
20 return a+b; }
21 //-----
22 int f (int a,int b,int c) {
23 cout << "-> B: i i i => i : " ;
24 return a+b+c; }
25 //-----
26 double f (double a,int b) {
27 cout << "-> C: d i => d : " ;
28 return a+b; }
```

18 f(1,2.9) -> A: i i => i : 3

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 int f (int, int ) ;
5 int f (int, int, int ) ;
6 double f (double, int ) ;
7 //int f (double, int ) ; // -> ΛΑΘΟΣ
8 //-----
9 int main(){
10 cout << "f(1,2) " << f(1,2) << endl;
11 cout << "f(1,2,3) " << f(1,2,3) << endl;
12 cout << "f(3.5,2) " << f(3.5,2) << endl;
13 cout << "f(1.9,2.9) " << f(1.9,2.9) << endl;
14 cout << "f(1,2.9) " << f(1,2.9) << endl;
15 cout << "f(1.9,2,3) " << f(1.9,2,3) << endl;
16 }
17 //-----
18 int f (int a,int b) {
19 cout << "-> A: i i => i : " ;
20 return a+b; }
21 //-----
22 int f (int a,int b,int c) {
23 cout << "-> B: i i i => i : " ;
24 return a+b+c; }
25 //-----
26 double f (double a,int b) {
27 cout << "-> C: d i => d : " ;
28 return a+b; }
```

20 f(1.9,2,3) -> B: i i i => i : 6

Συναρτήσεις

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 #include <iostream>
2 using namespace std;
3 //-----
4 int f (int, int ) ;
5 int f (int, int, int ) ;
6 double f (double, int ) ;
7 //int f (double, int ) ; // -> ΛΑΘΟΣ
8 //-----
9 int main(){
10 cout << "f(1,2) " << f(1,2) << endl;
11 cout << "f(1,2,3) " << f(1,2,3) << endl;
12 cout << "f(3.5,2) " << f(3.5,2) << endl;
13 cout << "f(1.9,2.9) " << f(1.9,2.9) << endl;
14 cout << "f(1,2.9) " << f(1,2.9) << endl;
15 cout << "f(1.9,2,3) " << f(1.9,2,3) << endl;
16 }
17 //-----
18 int f (int a,int b) {
19 cout << "-> A: i i => i : " ;
20 return a+b; }
21 //-----
22 int f (int a,int b,int c) {
23 cout << "-> B: i i i => i : " ;
24 return a+b+c; }
25 //-----
26 double f (double a,int b) {
27 cout << "-> C: d i => d : " ;
28 return a+b; }
```

```
10 f(1,2) -> A: i i => i : 3
12 f(1,2,3) -> B: i i i => i : 6
14 f(3.5,2) -> C: d i => d : 5.5
16 f(1.9,2.9) -> C: d i => d : 3.9
18 f(1,2.9) -> A: i i => i : 3
20 f(1.9,2,3) -> B: i i i => i : 6
```


- Μία μεταβλητή τύπου *δείκτη* (pointer) αποθηκεύει ως τιμή του τη *διεύθυνση* μίας μεταβλητής

- 1 Δήλωση ενός δείκτη προς μία μεταβλητή τύπου `char`. Ο τύπος του δείκτη είναι `char *`.

```
1 char *pc;  
2 char c = 'w';  
3 pc = &c;  
4 char d = *pc;
```

- Μία μεταβλητή τύπου *δείκτη* (pointer) αποθηκεύει ως τιμή του τη *διεύθυνση* μίας μεταβλητής

```
1 char *pc;  
2 char c = 'w';  
3 pc = &c;  
4 char d = *pc;
```

- 2 Ορισμός και αρχικοποίηση μίας μεταβλητής τύπου char.

- Μία μεταβλητή τύπου *δείκτη* (pointer) αποθηκεύει ως τιμή του τη *διεύθυνση* μίας μεταβλητής

```
1 char *pc;  
2 char c = 'w';  
3 pc = &c;  
4 char d = *pc;
```

- 3 Ο δείκτης λαμβάνει ως τιμή τη διεύθυνση της μεταβλητής c, με τη χρήση του τελεστή **&**.

- Μία μεταβλητή τύπου *δείκτη* (pointer) αποθηκεύει ως τιμή του τη *διεύθυνση* μίας μεταβλητής

```
1 char *pc;  
2 char c = 'w';  
3 pc = &c;  
4 char d = *pc;
```

- 4 Αποαναφοροποίηση (dereferencing) του δείκτη με τη χρήση του τελεστή *****. Η αποαναφοροποιημένη τιμή του δείκτη είναι ίση με την τιμή της μεταβλητής την οποία δείχνει. Εδώ π.χ. η μεταβλητή *d* ορίζεται ως τύπου *char* και αρχικοποιείται στην τιμή του *c*, μέσω της αποαναφοροποίησης **pc*.

- Μία μεταβλητή τύπου *δείκτη* (pointer) αποθηκεύει ως τιμή του τη *διεύθυνση* μίας μεταβλητής

```
1 char *pc;  
2 char c = 'w';  
3 pc = &c;  
4 char d = *pc;
```

- 1 Δήλωση ενός δείκτη προς μία μεταβλητή τύπου char. Ο τύπος του δείκτη είναι **char ***.
- 2 Ορισμός και αρχικοποίηση μίας μεταβλητής τύπου char.
- 3 Ο δείκτης λαμβάνει ως τιμή τη διεύθυνση της μεταβλητής c, με τη χρήση του τελεστή **&**.
- 4 Αποαναφοροποίηση (dereferencing) του δείκτη με τη χρήση του τελεστή *****. Η αποαναφοροποιημένη τιμή του δείκτη είναι ίση με την τιμή της μεταβλητής την οποία δείχνει. Εδώ π.χ. η μεταβλητή d ορίζεται ως τύπου char και αρχικοποιείται στην τιμή του c, μέσω της αποαναφοροποίησης *pc.

- Η διεύθυνση μνήμης μίας μεταβλητής λαμβάνεται με τη χρήση του τελεστή **&**.
- Ένας δείκτης που αποαναφοροποιείται με τη χρήση του τελεστή ***** ταυτίζεται με τη μεταβλητή στην οποία δείχνει.
- Ο τύπος δεδομένων του δείκτη οφείλει να ταιριάζει με τον τύπο των δεδομένων της μεταβλητής στην οποία δείχνει.
 - Π.χ. ένας δείκτης `char *` μπορεί να δείχνει μόνον σε μεταβλητές τύπου `char`.
 - Ο μεταγλωττιστής πρέπει να γνωρίζει το μέγεθος σε byte του αποαναφοροποιημένου τύπου δεδομένων.
- Υπάρχει ένας ειδικός τύπος δείκτη, ο `void *` (δείκτης προς το κενό \Rightarrow προς ο,τιδήποτε), ο οποίος μπορεί να δείχνει σε οποιαδήποτε θέση στη μνήμη, ανεξαρτήτως του τύπου δεδομένων που είναι εκεί αποθηκευμένα. Οι κενοί δείκτες δεν μπορούν να αποαναφοροποιηθούν.

Προσοχή: Δείκτες που δεν έχουν αρχικοποιηθεί αποτελούν την πιο συνηθισμένη αιτία αποτυχίας ενός προγράμματος κατά τη διάρκεια εκτέλεσής του.

- `int *pi;`

Δείκτης προς μία μεταβλητή int

- `double d=0.;`
`int *pi;`
`pi = &d; // Λάθος!!!`

ΛΑΘΟΣ στην τρίτη γραμμή: ο δείκτης pi είναι τύπου int *, οπότε δεν μπορεί να λάβει ως τιμή τη διεύθυνση της μεταβλητής d, αφού αυτή είναι τύπου double.

```
• const int i = 0;  
const int *pi = &i;  
*pi = 3; // Λάθος!!!
```

Ο δείκτης pi δηλώνεται ως εξής:

Ο pi είναι ένας δείκτης προς έναν ακέραιο που δεν μπορεί να μεταβληθεί (const int).

ΛΑΘΟΣ στην τρίτη γραμμή: Απαγορεύεται να μεταβάλλει κανείς έναν ακέραιο που δεν μπορεί να μεταβληθεί (const int).


```
• int i = 0;  
int j = 1;  
int * const pi = &i;  
pi = &j; // Λάθος!!!
```

Ο δείκτης `pi` δηλώνεται ως εξής:

Ο `pi` είναι ένας σταθερός δείκτης (`* const`) προς έναν ακέραιο (`int`) και αρχικοποιείται με τη διεύθυνση του `i`.

ΛΑΘΟΣ στην τέταρτη γραμμή: Ο `pi` απαγορεύεται να μεταβληθεί αφού έχει δηλωθεί σταθερός.

```
• int i = 0;  
  int *ri = &i;  
  int **rj;  
  rj = &ri;  
  **rj = 3;
```

Ο δείκτης rj δηλώνεται ως εξής:

Ο rj είναι δείκτης προς έναν δείκτη προς ακέραιο.

Ο rj παίρνει την τιμή της διεύθυνσης του δείκτη ri.

Άρα τώρα ο rj δείχνει εκεί όπου δείχνει ο ri, δηλαδή στην μεταβλητή i.

Έτσι, η τελευταία εντολή αποδίδει στη μεταβλητή i την τιμή 3.

Προσοχή: Η απόπειρα αποαναφοροποίησης τυχαίων θέσεων μνήμης μπορεί να οδηγήσει σε αποτυχία ενός προγράμματος κατά τη διάρκεια εκτέλεσής του.

```
• int *pi;  
int i = 0;  
if ( ... ) {  
    pi = &i; // εξαρτάται από το if, αν θα εκτελεστεί  
}  
*pi = 3;
```

Η εντολή `pi = &i;` ενδέχεται να μην εκτελεστεί, λόγω του `if`.
Τότε η τελευταία εντολή (`*pi = 3;`) μπορεί να οδηγήσει σε αποτυχία του προγράμματος κατά την εκτέλεσή του (segmentation fault).

Η θέση μνήμης 0x0 δεν είναι έγκυρη σε κανένα πρόγραμμα, και για το λόγο αυτό συνηθίζεται να χρησιμοποιείται για δείκτες που δεν έχουν κάποια συγκεκριμένη αρχική τιμή ή που είναι άκυροι.

Με την πρακτική αυτή, μπορεί κανείς να ελέγξει αν η τιμή ενός δείκτη είναι ίση με 0x0 πριν τον χρησιμοποιήσει.

```
• int *pi = 0; // Αρχικοποίηση στο 0x0  
int i = 0;  
if ( ... ) {  
    pi = &i; // εξαρτάται από το if, αν θα εκτελεστεί  
}  
if ( pi != NULL ) { // NULL = 0 εξ ορισμού  
    *pi = 3;  
}
```

Η εντολή `pi = &i;` ενδέχεται να μην εκτελεστεί, λόγω του `if`. Τότε η τελευταία εντολή (`*pi = 3;`) μπορεί να οδηγήσει σε αποτυχία του προγράμματος κατά την εκτέλεσή του (segmentation fault).

```
1 //-----
2 #include <iostream>
3 //-----
4 void fcn ( int * ) ;
5 //-----
6 using namespace std;
7 int main() {
8     int n=0;
9     cout << "n = " << n << endl;
10    fcn(&n);
11    cout << "n = " << n << endl;
12    return 0;
13 }
14 //-----
15 void fcn (int *pi) {
16     *pi = 3;
17 }
18 //-----
```

- 4 Η συνάρτηση fcn αναμένει ως μόνο όρισμά της έναν δείκτη προς έναν ακέραιο.

```
1 //-----  
2 #include <iostream>  
3 //-----  
4 void fcn ( int * ) ;  
5 //-----  
6 using namespace std;  
7 int main() {  
8     int n=0;  
9     cout << "n = " << n << endl;  
10    fcn(&n);  
11    cout << "n = " << n << endl;  
12    return 0;  
13 }  
14 //-----  
15 void fcn (int *pi) {  
16     *pi = 3;  
17 }  
18 //-----
```

8 Η μεταβλητή n αρχικοποιείται σε 0.

Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----  
2 #include <iostream>  
3 //-----  
4 void fcn ( int * ) ;  
5 //-----  
6 using namespace std;  
7 int main() {  
8     int n=0;  
9     cout << "n = " << n << endl;  
10    fcn(&n);  
11    cout << "n = " << n << endl;  
12    return 0;  
13 }  
14 //-----  
15 void fcn (int *pi) {  
16     *pi = 3;  
17 }  
18 //-----
```

9 Οθόνη: n = 0

```
1 //-----
2 #include <iostream>
3 //-----
4 void fcn ( int * ) ;
5 //-----
6 using namespace std;
7 int main() {
8     int n=0;
9     cout << "n = " << n << endl;
10    fcn(&n);
11    cout << "n = " << n << endl;
12    return 0;
13 }
14 //-----
15 void fcn (int *pi) {
16     *pi = 3;
17 }
18 //-----
```

- 10 Η συνάρτηση fcn καλείται με όρισμα τη διεύθυνση της μεταβλητής n.


```
1 //-----  
2 #include <iostream>  
3 //-----  
4 void fcn ( int * ) ;  
5 //-----  
6 using namespace std;  
7 int main() {  
8     int n=0;  
9     cout << "n = " << n << endl;  
10    fcn(&n);  
11    cout << "n = " << n << endl;  
12    return 0;  
13 }  
14 //-----  
15 void fcn (int *pi) {  
16     *pi = 3;  
17 }  
18 //-----
```

- 16 Η αποαναφοροποίηση του δείκτη pi, παρέχει πρόσβαση στην αρχική μεταβλητή n, η οποία έχει οριστεί εντός της main.

Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 //-----
4 void fcn ( int * ) ;
5 //-----
6 using namespace std;
7 int main() {
8     int n=0;
9     cout << "n = " << n << endl;
10    fcn(&n);
11    cout << "n = " << n << endl;
12    return 0;
13 }
14 //-----
15 void fcn (int *pi) {
16     *pi = 3;
17 }
18 //-----
```

11 Οθόνη: n = 3

Δείκτες

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 //-----
4 void fcn ( int * ) ;
5 //-----
6 using namespace std;
7 int main() {
8     int n=0;
9     cout << "n = " << n << endl;
10    fcn(&n);
11    cout << "n = " << n << endl;
12    return 0;
13 }
14 //-----
15 void fcn (int *pi) {
16     *pi = 3;
17 }
18 //-----
```

- 4 Η συνάρτηση fcn αναμένει ως μόνο όρισμά της έναν δείκτη προς έναν ακέραιο.
- 8 Η μεταβλητή n αρχικοποιείται σε 0.
- 9 Οθόνη: n = 0
- 10 Η συνάρτηση fcn καλείται με όρισμα τη διεύθυνση της μεταβλητής n.
- 16 Η αποαναφοροποίηση του δείκτη pi, παρέχει πρόσβαση στην αρχική μεταβλητή n, η οποία έχει οριστεί εντός της main.
- 11 Οθόνη: n = 3

Δείκτες

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 //-----
4 using namespace std;
5
6 int main() {
7
8     int i=100;
9     int j=333;
10
11     cout << "i = " << i
12          << ", j = " << j << endl;
13
14     // εδώ πρέπει να κληθεί η συνάρτησή σας
15
16     cout << "i = " << i
17          << ", j = " << j << endl;
18
19     return 0;
20 }
21 //-----
```

άσκηση

Να γραφεί ένα πρόγραμμα με μία συνάρτηση, την `swap`, η οποία να ανταλλάσσει μεταξύ τους τις τιμές δύο ακεραίων αριθμών. Δίνεται δίπλα ο πηγαίος κώδικας της συνάρτησης `main`.

Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

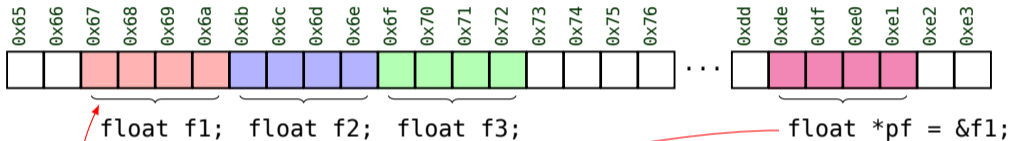
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

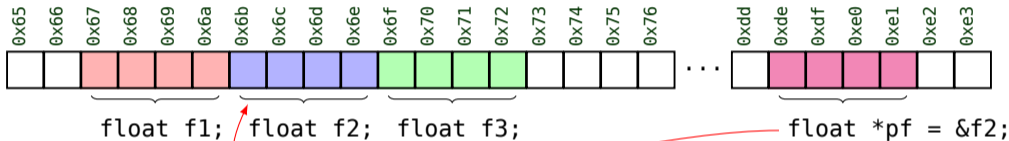
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

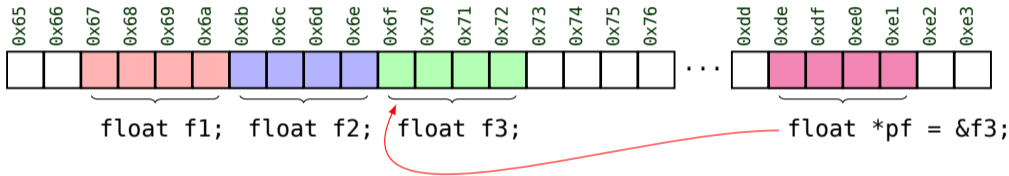
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

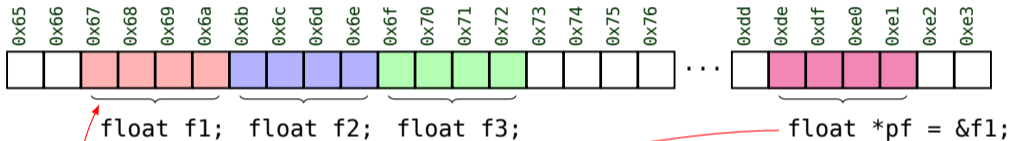
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

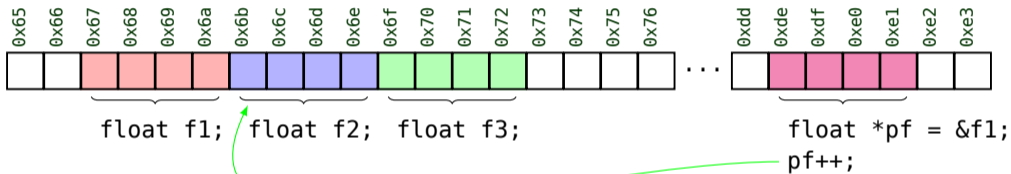
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

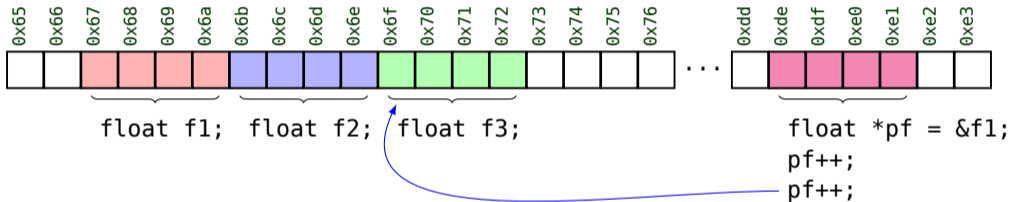
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

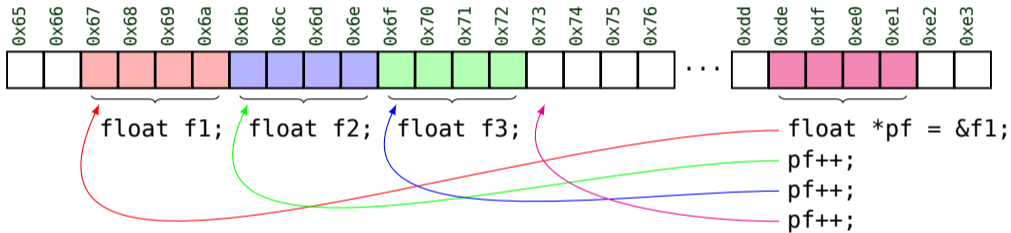
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

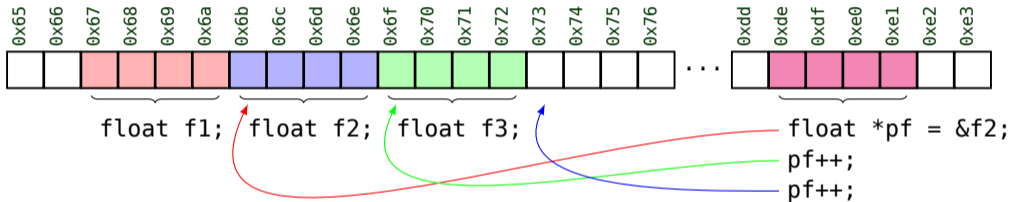
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

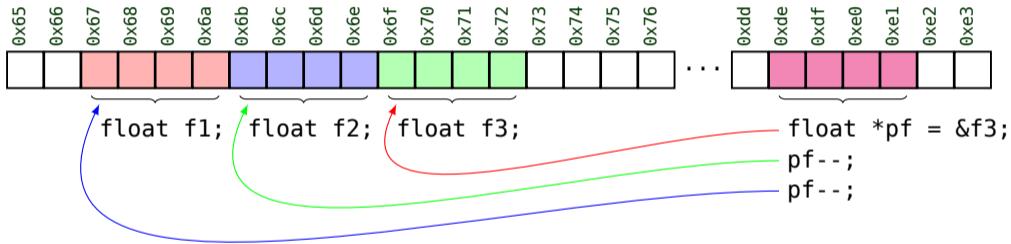
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Δείκτες

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

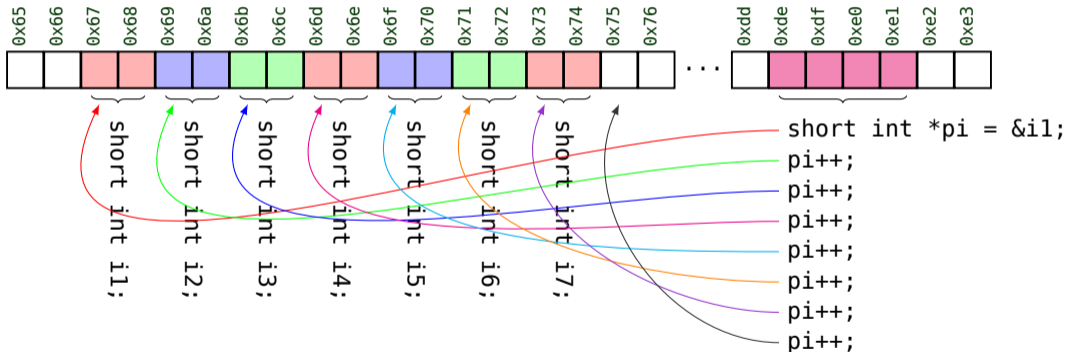
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Μνήμη



Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες
Μετατροπές
Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

- Μία ακολουθία αντικειμένων του ίδιου τύπου ονομάζεται *πίνακας*.
- Κάθε αντικείμενο του πίνακα αναφέρεται ως στοιχείο του πίνακα.
- Τα στοιχεία ενός πίνακα απασχολούν ένα συνεχές τμήμα της μνήμης, και είναι διατεταγμένα (πρώτο, δεύτερο, τρίτο στοιχείο κ.ο.κ.).

```
int a[10];           // Δήλωση ενός πίνακα ακεραίων με 10 στοιχεία
int b[3]={0,1,2};   // Δήλωση και αρχικοποίηση του πίνακα ακεραίων b με 3 στοιχεία
int c = b[0];       // Προσπέλαση του πρώτου στοιχείου του πίνακα b ( τελεστής [] )
b[0] = -1;
```

- Το πρώτο στοιχείο του πίνακα b είναι το b[0].

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
int *pb;           // Δήλωση ενός δείκτη προς ακέραιο int.
int b[3]={2,4,7}; // Δήλωση και αρχικοποίηση του πίνακα ακεραίων b με 3 στοιχεία
pb = &b[0];       // 0 pb δείχνει στην αρχή του πίνακα, στο πρώτο του στοιχείο
pb = b;          // Ισοδύναμη εντολή...
int c = *(pb+1);  // Προσπέλαση του στοιχείου b[1]
b[0] = -1;
```

- Το όνομα ενός πίνακα είναι ταυτόσημο με τη διεύθυνση του πρώτου του στοιχείου, εδώ του $b[0]$, δηλαδή $b \equiv \&b[0]$

- Ο δείκτης $pb+1$ δείχνει στο στοιχείο $b[1]$, κ.ο.κ., δηλαδή:

$pb \equiv \&b[0] \qquad *(pb) \equiv b[0]$

$pb+1 \equiv \&b[1] \qquad *(pb+1) \equiv b[1]$

$pb+2 \equiv \&b[2] \qquad *(pb+2) \equiv b[2]$

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 void f( const int * , int );
6 //-----
7 int main() {
8     int a[10];
9     int n = sizeof(a) / sizeof(a[0]);
10    for ( int i=0 ; i<n ; i++ ) {
11        a[i]=i*i;
12    }
13    f(a, n);
14    return 0;
15 }
16 //-----
17 void f( const int *apn, int an) {
18     for ( int j=0 ; j<an ; j++ ) {
19         cout << "a[" << j << " ] = " << *(apn+j) << endl;
20     }
21     return; // Η f επιστρέφει void, δηλαδή τίποτε...
22 }
23 //-----
```

- 5 Το πρώτο όρισμα της συνάρτησης f δεν θα πρέπει να μεταβληθεί. Έτσι δηλώνεται ως `const int *`.

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 void f( const int * , int );
6 //-----
7 int main() {
8     int a[10];
9     int n = sizeof(a) / sizeof(a[0]);
10    for ( int i=0 ; i<n ; i++ ) {
11        a[i]=i*i;
12    }
13    f(a, n);
14    return 0;
15 }
16 //-----
17 void f( const int *apn, int an) {
18     for ( int j=0 ; j<an ; j++ ) {
19         cout << "a[" << j << "] = " << *(apn+j) << endl;
20     }
21     return; // Η f επιστρέφει void, δηλαδή τίποτε...
22 }
23 //-----
```

- 9 Προσδιορισμός του πλήθους στοιχείων του πίνακα (συνάρτηση `sizeof`).

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 void f( const int * , int );
6 //-----
7 int main() {
8     int a[10];
9     int n = sizeof(a) / sizeof(a[0]);
10    for ( int i=0 ; i<n ; i++ ) {
11        a[i]=i*i;
12    }
13    f(a, n);
14    return 0;
15 }
16 //-----
17 void f( const int *apn, int an) {
18     for ( int j=0 ; j<an ; j++ ) {
19         cout << "a[" << j << "] = " << *(apn+j) << endl;
20     }
21     return; // Η f επιστρέφει void, δηλαδή τίποτε...
22 }
23 //-----
```

10 Ο πίνακας γεμίζει με αριθμούς.

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 void f( const int * , int );
6 //-----
7 int main() {
8     int a[10];
9     int n = sizeof(a) / sizeof(a[0]);
10    for ( int i=0 ; i<n ; i++ ) {
11        a[i]=i*i;
12    }
13    f(a, n);
14    return 0;
15 }
16 //-----
17 void f( const int *apn, int an) {
18     for ( int j=0 ; j<an ; j++ ) {
19         cout << "a[" << j << " ] = " << *(apn+j) << endl;
20     }
21     return; // Η f επιστρέφει void, δηλαδή τίποτε...
22 }
23 //-----
```

- 13 Κλήση της συνάρτησης f με πρώτο όρισμα τον δείκτη apn προς το πρώτο στοιχείο του πίνακα και δεύτερο όρισμα το πλήθος στοιχείων του πίνακα.

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 void f( const int * , int );
6 //-----
7 int main() {
8     int a[10];
9     int n = sizeof(a) / sizeof(a[0]);
10    for ( int i=0 ; i<n ; i++ ) {
11        a[i]=i*i;
12    }
13    f(a, n);
14    return 0;
15 }
16 //-----
17 void f( const int *apn, int an) {
18     for ( int j=0 ; j<an ; j++ ) {
19         cout << "a[" << j << "] = " << *(apn+j) << endl;
20     }
21     return; // Η f επιστρέφει void, δηλαδή τίποτε...
22 }
23 //-----
```

- 18 Η συνάρτηση f εμφανίζει στην οθόνη τις τιμές όλων των στοιχείων του πίνακα, ξεκινώντας από το την τιμή του ακεραίου που δείχνει ο δείκτης apn.

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 void f( const int * , int );
6 //-----
7 int main() {
8     int a[10];
9     int n = sizeof(a) / sizeof(a[0]);
10    for ( int i=0 ; i<n ; i++ ) {
11        a[i]=i*i;
12    }
13    f(a, n);
14    return 0;
15 }
16 //-----
17 void f( const int *apn, int an) {
18     for ( int j=0 ; j<an ; j++ ) {
19         cout << "a[" << j << " ] = " << *(apn+j) << endl;
20     }
21     return; // Η f επιστρέφει void, δηλαδή τίποτε...
22 }
23 //-----
```

19 Τιμή του j-οστού ακεραίου.

Πίνακες (Arrays)

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 void f( const int * , int );
6 //-----
7 int main() {
8     int a[10];
9     int n = sizeof(a) / sizeof(a[0]);
10    for ( int i=0 ; i<n ; i++ ) {
11        a[i]=i*i;
12    }
13    f(a, n);
14    return 0;
15 }
16 //-----
17 void f( const int *apn, int an) {
18     for ( int j=0 ; j<an ; j++ ) {
19         cout << "a[" << j << "]" = " << *(apn+j) << endl;
20     }
21     return; // Η f επιστρέφει void, δηλαδή τίποτε...
22 }
23 //-----
```

- 5 Το πρώτο όρισμα της συνάρτησης f δεν θα πρέπει να μεταβληθεί. Έτσι δηλώνεται ως **const int ***.
- 9 Προσδιορισμός του πλήθους στοιχείων του πίνακα (συνάρτηση **sizeof**).
- 10 Ο πίνακας γεμίζει με αριθμούς.
- 13 Κλήση της συνάρτησης f με πρώτο όρισμα τον δείκτη apn προς το πρώτο στοιχείο του πίνακα και δεύτερο όρισμα το πλήθος στοιχείων του πίνακα.
- 18 Η συνάρτηση f εμφανίζει στην οθόνη τις τιμές όλων των στοιχείων του πίνακα, ξεκινώντας από το την τιμή του ακεραίου που δείχνει ο δείκτης apn.
- 19 Τιμή του j-οστού ακεραίου.

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Ένα αλφαριθμητικό της C (C-string) είναι ένας πίνακας χαρακτήρων, με τον τελευταίο χαρακτήρα να είναι ο `'\0'` (NUL).

Ο χαρακτήρας αυτός χρησιμοποιείται συχνά σε συναρτήσεις που υπολογίζουν το μήκος C-strings.

Στη C αυτά τα C-strings αποτελούν τον βασικό τύπο αποθήκευσης κειμένου.

Στη C++ έχουν σε μεγάλο βαθμό αντικατασταθεί από την κλάση *string*.

- 7 Ο μεταγλωττιστής μπορεί να προσδιορίσει αυτόματα το μήκος ενός πίνακα κατά την αρχικοποίησή του.

```
char str[] = "hello world"; // ή ισοδύναμα:  
char str[] = {'h','e','l','l','o',  
              ' ','w','o','r','l','d','\0'}
```

```
1 //-----  
2 #include <iostream>  
3 using namespace std;  
4 //-----  
5 int main(){  
6     char str[]="hello world";  
7  
8  
9     int n = sizeof(str) / sizeof(char);  
10  
11     cout << str << " is " << n << " char long" << endl;  
12  
13     for ( int i=0 ; i<n ; i++ ) {  
14         int c = str[i];  
15         cout << "ASCII " << str[i] << ": " << c << endl;  
16     }  
17  
18     return 0;  
19  
20 }  
21 //-----
```

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Ένα αλφαριθμητικό της C (C-string) είναι ένας πίνακας χαρακτήρων, με τον τελευταίο χαρακτήρα να είναι ο `'\0'` (NUL).

Ο χαρακτήρας αυτός χρησιμοποιείται συχνά σε συναρτήσεις που υπολογίζουν το μήκος C-strings.

Στη C αυτά τα C-strings αποτελούν τον βασικό τύπο αποθήκευσης κειμένου.

Στη C++ έχουν σε μεγάλο βαθμό αντικατασταθεί από την κλάση *string*.

- 7 Ο μεταγλωττιστής μπορεί να προσδιορίσει αυτόματα το μήκος ενός πίνακα κατά την αρχικοποίησή του.

```
char str[] = "hello world"; // ή ισοδύναμα:  
char str[] = {'h','e','l','l','o',  
              ' ','w','o','r','l','d','\0'}
```

- 9 Προσδιορισμός του πλήθους στοιχείων του πίνακα (συνάρτηση **sizeof**).

```
1 //-----  
2 #include <iostream>  
3 using namespace std;  
4 //-----  
5 int main(){  
6     char str[]="hello world";  
7  
8  
9     int n = sizeof(str) / sizeof(char);  
10  
11     cout << str << " is " << n << " char long" << endl;  
12  
13     for ( int i=0 ; i<n ; i++ ) {  
14         int c = str[i];  
15         cout << "ASCII " << str[i] << ": " << c << endl;  
16     }  
17  
18     return 0;  
19  
20 }  
21 //-----
```

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Ένα αλφαριθμητικό της C (C-string) είναι ένας πίνακας χαρακτήρων, με τον τελευταίο χαρακτήρα να είναι ο `'\0'` (NUL).

Ο χαρακτήρας αυτός χρησιμοποιείται συχνά σε συναρτήσεις που υπολογίζουν το μήκος C-strings.

Στη C αυτά τα C-strings αποτελούν τον βασικό τύπο αποθήκευσης κειμένου.

Στη C++ έχουν σε μεγάλο βαθμό αντικατασταθεί από την κλάση *string*.

- 7 Ο μεταγλωττιστής μπορεί να προσδιορίσει αυτόματα το μήκος ενός πίνακα κατά την αρχικοποίησή του.

```
char str[] = "hello world"; // ή ισοδύναμα:  
char str[] = {'h','e','l','l','o',  
              ' ','w','o','r','l','d','\0'}
```

- 9 Προσδιορισμός του πλήθους στοιχείων του πίνακα (συνάρτηση **sizeof**).

- 11 Οθόνη: hello world is 12 char long

```
1 //-----  
2 #include <iostream>  
3 using namespace std;  
4 //-----  
5 int main(){  
6     char str[]="hello world";  
7  
8  
9     int n = sizeof(str) / sizeof(char);  
10  
11     cout << str << " is " << n << " char long" << endl;  
12  
13     for ( int i=0 ; i<n ; i++ ) {  
14         int c = str[i];  
15         cout << "ASCII " << str[i] << ": " << c << endl;  
16     }  
17  
18     return 0;  
19  
20 }  
21 //-----
```

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Ένα αλφαριθμητικό της C (C-string) είναι ένας πίνακας χαρακτήρων, με τον τελευταίο χαρακτήρα να είναι ο `'\0'` (NUL).

Ο χαρακτήρας αυτός χρησιμοποιείται συχνά σε συναρτήσεις που υπολογίζουν το μήκος C-strings.

Στη C αυτά τα C-strings αποτελούν τον βασικό τύπο αποθήκευσης κειμένου.

Στη C++ έχουν σε μεγάλο βαθμό αντικατασταθεί από την κλάση *string*.

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 int main(){
6     char str[]="hello world";
7
8     int n = sizeof(str) / sizeof(char);
9
10    cout << str << " is " << n << " char long" << endl;
11
12    for ( int i=0 ; i<n ; i++ ) {
13        int c = str[i];
14        cout << "ASCII " << str[i] << ": " << c << endl;
15    }
16
17    return 0;
18
19 }
20
21 //-----
```

- 7 Ο μεταγλωττιστής μπορεί να προσδιορίσει αυτόματα το μήκος ενός πίνακα κατά την αρχικοποίησή του.

```
char str[] = "hello world"; // ή ισοδύναμα:
char str[] = {'h','e','l','l','o',
              ' ','w','o','r','l','d','\0'}
```

- 9 Προσδιορισμός του πλήθους στοιχείων του πίνακα (συνάρτηση **sizeof**).
- 11 Οθόνη: hello world is 12 char long
- 14 Το c παίρνει την τιμή του i-οστού χαρακτήρα του str.

Πίνακες (Arrays)

C++ / ROOT

I. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

Ένα αλφαριθμητικό της C (C-string) είναι ένας πίνακας χαρακτήρων, με τον τελευταίο χαρακτήρα να είναι ο `'\0'` (NUL).

Ο χαρακτήρας αυτός χρησιμοποιείται συχνά σε συναρτήσεις που υπολογίζουν το μήκος C-strings.

Στη C αυτά τα C-strings αποτελούν τον βασικό τύπο αποθήκευσης κειμένου.

Στη C++ έχουν σε μεγάλο βαθμό αντικατασταθεί από την κλάση *string*.

```
1 //-----
2 #include <iostream>
3 using namespace std;
4 //-----
5 int main(){
6     char str[]="hello world";
7
8     int n = sizeof(str) / sizeof(char);
9
10    cout << str << " is " << n << " char long" << endl;
11
12    for ( int i=0 ; i<n ; i++ ) {
13        int c = str[i];
14        cout << "ASCII " << str[i] << " : " << c << endl;
15    }
16
17    return 0;
18
19 }
20
21 //-----
```

- 7 Ο μεταγλωττιστής μπορεί να προσδιορίσει αυτόματα το μήκος ενός πίνακα κατά την αρχικοποίησή του.

```
char str[] = "hello world"; // ή ισοδύναμα:
char str[] = {'h','e','l','l','o',
              ' ','w','o','r','l','d','\0'}
```

- 9 Προσδιορισμός του πλήθους στοιχείων του πίνακα (συνάρτηση **sizeof**).
- 11 Οθόνη: hello world is 12 char long
- 14 Το c παίρνει την τιμή του i-οστού χαρακτήρα του str.
- 15 Οθόνη:

```
ASCII h: 104
ASCII e: 101
ASCII l: 108
ASCII l: 108
ASCII o: 111
ASCII  : 32
ASCII w: 119
ASCII o: 111
ASCII r: 114
ASCII l: 108
ASCII d: 100
ASCII  : 0
```

Σημείωση: Ο χαρακτήρας `'\0'` δεν μπορεί να τυπωθεί στην οθόνη.

Πίνακες (Arrays)

C++ / ROOT

Ι. Παπαδόπουλος

Περιεχόμενα

Υπονοούμενες

Μετατροπές

Τύπων

Συναρτήσεις

Δείκτες

Πίνακες

```
char k[3][5];
```

στήλες

	0	1	2	3	4
0	k[0][0]	k[0][1]	k[0][2]	k[0][3]	k[0][4]
1	k[1][0]	k[1][1]	k[1][2]	k[1][3]	k[1][4]
2	k[2][0]	k[2][1]	k[2][2]	k[2][3]	k[2][4]

γραμμές

Μνήμη

