



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 5: Η ΓΛΩΣΣΑ C++

Εισαγωγή στην C++

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Η Γλώσσα C++

ΙΣΤΟΡΙΑ

1967: **Simula67** (Νορβηγία) -> πρώτη αντικειμενοστρεφής γλώσσα

'70: **Smalltalk** (Palo Alto, CA) -> κάθε στοιχείο ένα αντικείμενο
Αρχές '80: ο αντικειμενοστρεφής τρόπος σκέψης εισάγεται σε ακαδημαϊκούς κύκλους

'80: **C++** (Stroustrup, AT&T): σοβαρή, αποδοτική γλώσσα, πρότυπο στη βιομηχανία

1995: **Java** (Sun Microsystems)



ΔΙΑΦΟΡΕΣ ΜΕ C

C

- Διαδικαστικός προγραμματισμός
- Από άνω προς τα κάτω, σχεδίαση προγράμματος με C

C++

- Object based προγραμματισμός (κλάσεις, αντικείμενα, ενθυλάκωση)
- Object oriented προγραμματισμός (κληρονομικότητα, πολυμορφισμός)
- Generic πολυμορφισμός (class και function templates)



COMPILERS

- CYGWIN (G++)
- Devc++
- Code::blocks



ΕΙΣΟΔΟΣ ΕΞΟΔΟΣ

■ **cin**

- Standard input stream
- Συνήθως από το πληκτρολόγιο

■ **cout**

- Standard output stream
- Συνήθως έξοδος στην οθόνη

■ **cerr**

- Standard error stream
- Προβολή μηνυμάτων σφαλμάτων



Απλό πρόγραμμα - Εκτύπωση

```
#include <iostream>
int main() {
    std::cout << "Hello World!\n";
    return 0;
}
```

Hello World!

`std::cout` είναι το κανάλι εξόδου στην οθόνη

Χρήση τελεστή `<<`

Χαρακτήρας escape: `/`

`/n` επόμενη γραμμή

`/t` κενό "tab"

`//` χαρακτήρας `/`

`/"` χαρακτήρας `"`

Παρατήρηση: Δεν είναι ανάγκη να βρίσκεται μέσα σε κάποια κλάση όπως στην Java



Απλό πρόγραμμα – Είσοδος/Εξοδος

```
#include <iostream>
int main() {
    int num; std::cout << "Enter integer\n";
    std::cin >> num;
    std::cout << "Your number was: " << num << std::endl;
    return 0;
}
```

```
Enter integer
5
Your number was: 5
```

Διαδοχική χρήση του τελεστή <<

`std::in` είναι το κανάλι εισόδου από το πληκτρολόγιο

Χρήση τελεστή >>

Το `std::endl` μεταφέρει σε επόμενη γραμμή (αναγκάζει το κείμενο να εκτυπωθεί)



Χρήση Using, Inline συναρτήσεις

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

inline double diameter( const double r ) { return 2 * r; }

int main(){
    double r;
    for ( int k = 1; k < 3; k++ ) {
        cout << "Enter radius of circle: ";
        cin >> r;
        cout << "The diameter of the circle is "
             << diameter( r ) << endl;
    }
    return 0;
}
```

- Το `std::` στην αρχη μπορεί να παραληφθεί αν έχει γίνει πριν η δήλωση `using std::cout;`
- Συναρτήσεις με τον προσδιοριστή `inline`, “αναγκάζουν” τον compiler να ενσωματώσει τον κώδικα τους στο σημείο όπου καλούνται.
(καλύτερη απόδοση σε ταχύτητα εκτέλεσης, αλλά αυξάνεται το μέγεθος του εκτελέσιμου)



Πέρασμα με Αναφορά

```
#include <iostream>
using std::cout;
using std::endl;

int fooByValue ( int x) {
    x = x + 5;
    return x;
}

void fooByReference ( int &x ) {
    x = x + 5;
}

int main() {
    int number = 4;
    cout << "fooByValue returns: " << fooByValue(number) << endl;
    cout << "Number: " << number << endl; fooByReference(number);
    cout << "Number" << number << endl;
    return 0;
}
```

Περνάει στην συνάρτηση ένα αντίγραφο του ορίσματος (δεν αλλάζει η τιμή του number)

Περνάει το ίδιο το όρισμα (αύξηση του x σημαίνει αύξηση του number)

fooByValue returns: 9

Number: 4

Number9

Δεν άλλαξε η τιμή του **number** από την κληση της **fooByValue**

Άλλαξε η τιμή του **number** από την κληση της **fooByReference**



Προκαθορισμένες Τιμές Ορισμάτων

Δήλωση προκαθορισμένων τιμών για τα ορίσματα συνάρτησης για την περίπτωση που παραληφθούν.

```
#include <iostream>
using std::cout;
using std::endl;

int foo( int a = 0, int b = 0, int c = 0 ){
    return a*100 + b* 10 + c; }

int main(){
    cout << "All default: " << foo() <<endl
         << "b,c default: " << foo( 1 ) <<endl
         << "c default: " << foo( 1,2 ) <<endl
         << "no default: " << foo( 1, 2, 3 ) <<endl;
    return 0;
}
```

```
All default: 0
b,c default: 100
c default: 120
no default: 123
```



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 2** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 5: Η ΓΛΩΣΣΑ C++

Δομές Ελέγχου

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος
Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Δομές Ελέγχου

Εισαγωγή

- Πριν γράψουμε ένα πρόγραμμα
 - Κατανοούμε πλήρως το πρόβλημα
 - Σχεδιάζουμε προσεκτικά την παρεχόμενη λύση
- Όσο γράφουμε το πρόγραμμα
 - Γνωρίζουμε ποια δομικά μέρη είναι διαθέσιμα
 - Χρησιμοποιούμε σωστές αρχές προγραμματισμού

The art of computer programming του Donald E. Knuth



Λέξεις Κλειδιά

Κοινά με την C

Auto	break	case	char	const
Continue	default	do	double	else
Enum	extern	float	for	goto
If	int	long	register	return
Short	signed	sizeof	static	struct
Switch	typedef	union	unsigned	void
Volatile	while			

Μόνο στην C++

Asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	true
static_cast	template	this	throw	virtual
try	typeid	typename	using	
wchar_t				

Casting

```
#include <iostream>
#include <iomanip>
using std::cout;
using std::endl;
using std::fixed;
using std::setprecision;
int main() {
    int totalAge = 75;
    int numPeople = 4;
    double averageAge = totalAge / numPeople;
    double averageAge2 = static_cast< double >( totalAge ) / numPeople;

    cout << "Avarage age is " << setprecision( 2 ) << fixed << averageAge << endl
         << "Avarage age2 is " << setprecision( 2 ) << fixed << averageAge2 << endl;
    return 0;
}
```

Η `fixed` αναγκάζει την έξοδο να εκτυπωθεί σε μορφή σταθερής υποδιαστολής (όχι σε επιστημονική μορφή). Επίσης, αναγκάζει να τυπώνονται η υποδιαστολή ακολουθούμενη από απαραίτητα μηδενικά.

Η `setprecision(2)` εκτυπώνει δύο ψηφία μετά την υποδιαστολή (στρογγυλοποίηση για να ταιριάζει στην ακρίβεια). Βιβλιοθήκη: `<iomanip>`

```
Avarage age is 18.00
Avarage age2 is 18.75
```

Η `static_cast<double>()` χειρίζεται το `totalAge` ως `double` πρόσκαιρα (casting). Απαιτείται διότι η διαίρεση δύο ακεραίων αποκόβει το υπόλοιπο.



Τελεστές Ανάθεσης

μεταβλητή τελεστής = έκφραση;
(μεταβλητή = μεταβλητή τελεστής έκφραση;)

d -= 4	(d = d - 4)
e *= 5	(e = e * 5)
f /= 3	(f = f / 3)
g %= 9	(g = g % 9)



Τελεστές αύξησης και μείωσης

- Ο τελεστής αύξησης ($++$) – μπορεί να αντικαταστήσει το: $c += 1$
- Ο τελεστής μείωσης ($--$) - μπορεί να αντικαταστήσει το: $c -= 1$
 - Προ-αύξηση
 - Όταν ο τελεστής χρησιμοποιείται πριν τη μεταβλητή ($++c$ ή $--c$)
 - Η μεταβλητή αλλάζει και στη συνέχεια υπολογίζεται η έκφραση στην οποία περιλαμβάνεται
 - Μετά-αύξηση
 - Όταν ο τελεστής χρησιμοποιείται μετά τη μεταβλητή ($c++$ ή $c--$)
 - Εκτελείται η έκφραση στην οποία περιλαμβάνεται η μεταβλητή και στη συνέχεια η μεταβλητή λαμβάνει τη νέα τιμή.



Τελεστές αύξησης και μείωσης

```
#include <iostream>
using std::cout;
using std::endl;

int main(){
    int x= 1;
    cout << x; // 1
    cout << x++; // 1 Πρώτα εκτύπωση μετά αύξηση
    cout << x << endl; // 2

    x = 1;
    cout << x; // 1
    cout << --x; // 0 Πρώτα αφαίρεση μετά εκτύπωση
    cout << x; // 0
    return 0;
}
```

```
112
100
```



Δομή Ελέγχου switch

```
#include <iostream>
using std::cout;
using std::cin;
int main(){
    int num;
    std::cin >> num;
    switch ( num ) {
        case 1: cout << " value 1";
            break;
        case 2:
        case 3:
            cout << " value 2 or 3";
            break;
        case (2+2):
            cout << " value 4";
        default:
            cout << " - other value.";
    }
    return 0;
}
```

Με το break συνεχίζει μετά το τέλος του switch. Το βάζουμε στο τέλος κάθε περίπτωσης ώστε να μην ελέγξει και τις επόμενες.

1
value 1

4
value 4 - other value



do/while δομή επανάληψης

- Όμοια με τη δομή **while**
 - Δημιουργεί ένα βρόγχο που ελέγχεται στο τέλος και όχι στην αρχή
 - Οι δηλώσεις στο σώμα εκτελούνται τουλάχιστον μία φορά
- Μορφή

```
do {  
    δήλωση  
} while ( συνθήκη );
```



break και continue

- Δήλωση: **break**
 - Άμεση έξοδο από **while**, **for**, **do/while**, **switch**
 - Το πρόγραμμα συνεχίζει με την πρώτη δήλωση μετά τη δομή
- Χρησιμοποιείται σε περιπτώσεις όπως:
 - Νωρίτερη διαφυγή από το βρόγχο
 - Παράβλεψη του υπόλοιπου **switch**



break και continue

- **Δήλωση: `continue`**
 - Χρησιμοποιείται στις `while`, `for`, `do/while`
 - Διαφυγή από το υπόλοιπο βρόγχο
 - Προχωρά στην επόμενη επανάληψη του βρόγχου
- **`while` και `do/while` δομές**
 - Ο έλεγχος για συνέχεια της επανάληψης ελέγχεται αμέσως μετά τη δήλωση `continue`
- **`for` δομή**
 - Εκτελείται αύξηση της έκφρασης
 - Στη συνέχεια, ελέγχεται αν θα συνεχιστεί ο βρόγχος



Παράδειγμα continue

```
for ( int x = 1; x <= 10; x++ ) {  
    if ( x % 2 )  
        continue;  
    cout << x << " ";  
}
```

2 4 6 8 10



Τελεστές Σύγκρισης (στις συνθήκες)

- $>$ μεγαλύτερο
- $<$ μικρότερο
- $>=$ μεγαλύτερο ίσο
- $<=$ μικρότερο ίσο
- $==$ ίδια τιμή
- $!=$ διαφορετική τιμή



Λογικοί Τελεστές

- Χρησιμοποιούνται ως συνθήκες σε βρόγχους και δηλώσεις `if`

- **&&** (λογικό **ΚΑΙ**)

- **true** αν και οι δύο συνθήκες είναι **true**

```
if ( gender == 1 && age >= 65 )  
    ++seniorFemales;
```

- **||** (λογικό **Ή**)

- **true** αν μία από τις δύο συνθήκες είναι **true**

```
if ( semesterAverage >= 90 || finalExam >= 90 )  
    cout << "Student grade is A" << endl;
```



Λογικοί Τελεστές

- Χρησιμοποιούνται ως συνθήκες σε βρόγχους και δηλώσεις `if`

- **&&** (λογικό **ΚΑΙ**)

- **true** αν και οι δύο συνθήκες είναι **true**

```
if ( gender == 1 && age >= 65 )  
    ++seniorFemales;
```

- **||** (λογικό **Ή**)

- **true** αν μία από τις δύο συνθήκες είναι **true**

```
if ( semesterAverage >= 90 || finalExam >= 90 )  
    cout << "Student grade is A" << endl;
```



Λογικοί Τελεστές

- ! (λογικό NOT)

- Επιστρέφει **true** όταν η συνθήκη είναι **false**, & αντιστρόφως

```
if ( !( grade == sentinelValue ) )  
    cout << "The next grade is " <<  
grade << endl;
```

Εναλλακτικά:

```
if ( grade != sentinelValue )  
    cout << "The next grade is " <<  
grade << endl;
```



Ισότητα (==) και Ανάθεση (=)

- Κοινό λάθος
 - Δεν επιστρέφεται συνήθως συντακτικό λάθος
- Όψεις του προβλήματος
 - Οι εκφράσεις που έχουν τιμή μπορούν να χρησιμοποιηθούν για να ληφθεί απόφαση
 - Zero = false, nonzero = true
 - Οι δηλώσεις ανάθεσης παράγουν μία τιμή (αυτή που αναθέτουν)



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαια 4, 5** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_htp_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 5: Η ΓΛΩΣΣΑ C++
Συναρτήσεις - Μεταβλητές

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος
Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Συναρτήσεις / Μεταβλητές

Μαθηματικές Συναρτήσεις

- ... Υλοποίηση απλών μαθηματικών λειτουργιών
 - Απαιτείται η συμπερίληψη του header file `<cmath>`
- Παράδειγμα

```
cout << sqrt( 900.0 );
```

- Το αποτέλεσμα είναι η εκτύπωση της τιμής 30.0
- Όλες οι συναρτήσεις της math library επιστρέφουν **double**



Μαθηματικές Συναρτήσεις

Μέθοδος	Περιγραφή
<code>ceil(x)</code>	Στρογγυλοποίηση προς το πάνω (πλησιέστερο ακέραιο)
<code>cos(x)</code>	Συνημίτονο (x σε radians)
<code>exp(x)</code>	Exponential e^x
<code>fabs(x)</code>	Απόλυτη τιμή
<code>floor(x)</code>	Στρογγυλοποίηση προς τα κάτω (πλησιέστερος ακέραιος)
<code>fmod(x, y)</code>	Υπόλοιπο του x/y ως κινητής υποδιαστολής αριθμός
<code>log(x)</code>	Φυσικός Λογάριθμος του x (βάση το e)
<code>log10(x)</code>	Λογάριθμος του x (βάση το 10)
<code>pow(x, y)</code>	Ύψωση του x στην y (x^y)
<code>sin(x)</code>	Ημίτονο (x σε radians)
<code>sqrt(x)</code>	Τετραγωνική ρίζα
<code>tan(x)</code>	Εφαπτομένη (x σε radians)



Ορισμοί Συναρτήσεων

- Πρωτότυπο συνάρτησης
 - Ενημερώνει το μεταγλωττιστή για τον τύπο των παραμέτρων και τον επιστρεφόμενο τύπο της συνάρτησης
 - `int square(int);`
 - Συνάρτηση που λαμβάνει `int` και επιστρέφει `int`
- Κλήση συνάρτησης
 - `square(x);`
 - Παρενθέσεις, τελεστής κλήσης συνάρτησης
 - Πέρασμα παραμέτρου `x`
 - Η συνάρτηση δέχεται το δικό της αντίγραφο των παραμέτρων
 - Αφού τερματίσει περνάει πίσω το αποτέλεσμα



Πρωτότυπα συναρτήσεων

- Αρχικές δηλώσεις των συναρτήσεων ώστε να μπορούν χρησιμοποιηθούν από πρόγραμμα
- Το πρωτότυπο πρέπει να ταιριάζει με τον ορισμό της συνάρτησης

- Function prototype

```
double maximum( double, double, double );
```

- Definition

```
double maximum( double x, double y, double z ){  
    ...  
}
```

- Υπογραφή συνάρτησης

- Το μέρος του πρωτοτύπου με όνομα και παραμέτρους

- `double maximum(double, double, double);`



Header Files (Αρχεία επικεφαλίδων)

- Περιέχουν
 - Πρωτότυπα συναρτήσεων
 - Ορισμούς τύπων και σταθερών
- Τα αρχεία επικεφαλίδων έχουν κατάληξη .h
 - Programmer-defined header files

```
#include "myheader.h"
```
- Library header files

```
#include <cmath>
```



Γεννήτρια τυχαίων αριθμών

- **rand** function (`<cstdlib>`)

- `i = rand();`

- Παράγει έναν unsigned integer μεταξύ 0 και RAND_MAX (συνήθως 32767)

- Scaling and shifting

- Παράδειγμα

- `i = rand() % 6 + 1;`

- “`Rand() % 6`” παράγει έναν αριθμό μεταξύ 0 και 5 (scaling)
 - “`+ 1`” δίνει το διάστημα 1 έως 6 (shift)



Παράδειγμα

```
for ( int counter = 1; counter <= 20; counter++ )  
    cout << setw( 10 ) << ( 1 + rand() % 6 );
```

Παραγωγή και εκτύπωση 20 τυχαίων αριθμών από 1 έως 6.



Γεννήτρια τυχαίων αριθμών

- Η χρήση της `rand()` σε επαναλαμβανόμενες εκτελέσεις του προγράμματος
 - Αποδίδει την ίδια ακολουθία αριθμών
- Για να πάρουμε διαφορετικές ακολουθίες αριθμών
 - Χρησιμοποιούμε μια τιμή φύτρο (`seed value`)
 - Αντιστοιχεί σε τυχαίο σημείο εκκίνησης της ακολουθίας
 - Το ίδιο `seed` θα αποδώσει την ίδια ακολουθία
 - **`srand (seed) ;`**
 - Βιβλιοθήκη: `<cstdlib>`
 - Χρησιμοποιείται πριν από τη `rand ()` για να προσδιορίσει το τυχαίο σημείο εκκίνησης



Γεννήτρια τυχαίων αριθμών

- Μπορούμε να χρησιμοποιήσουμε την τρέχουσα ώρα ως φυτό (seed)
 - `srand(time(0));`
 - `time(0);`
 - `<ctime>`
 - Επιστρέφει την τρέχουσα ώρα σε δευτερόλεπτα



Χαρακτηριστικά Μεταβλητών

- Οι μεταβλητές χαρακτηρίζονται από διάφορες ιδιότητες
 - όνομα, τύπο δεδομένων, μέγεθος, τιμή
 - Τάξη αποθήκευσης
 - Για ποιο διάστημα η μεταβλητή υπάρχει στη μνήμη
 - Εμβέλεια (Scope)
 - Το τμήμα του προγράμματος που μια αναφορά στη μεταβλητή είναι έγκυρη
 - Συνδεσιμότητα (Linkage)
 - Όταν ένα πρόγραμμα υλοποιείται σε πολλά αρχεία, ποιά αρχεία μπορούν να τη χρησιμοποιήσουν



Τάξη αποθήκευσης

- **Automatic**

Η μεταβλητή δημιουργείται όταν ο έλεγχος εισέρχεται σ' ένα block εντολών και καταστρέφεται όταν ο έλεγχος εξέρχεται από το block εντολών

- **Static**

Οι μεταβλητές εξακολουθούν να υπάρχουν καθ' όλη τη διάρκεια ζωής του προγράμματος

- **Extern**

Η κατάσταση της μεταβλητής είναι γνωστή σε κάθε συνάρτηση που ακολουθεί



Κανόνες Εμβέλειας

- **Εμβέλεια (Scope)**
 - Η περιοχή του προγράμματος που ένα όνομα (π.χ. μεταβλητής) μπορεί να χρησιμοποιηθεί
- **Εμβέλεια αρχείου (File scope)**
 - Ορίζεται εκτός μιας συνάρτησης, διαθεσιμότητα για όλες τις συναρτήσεις
 - Σφαιρικές μεταβλητές, οι ορισμοί και τα πρωτότυπα συναρτήσεων
- **Εμβέλεια συνάρτησης (Function scope)**
 - Η περιοχή πρόσβασης περιορίζεται μόνο στη συνάρτηση που ορίζεται το όνομα



Κανόνες Εμβέλειας

- **Εμβέλεια μπλοκ (Block scope)**
 - Ξεκινά από το σημείο δήλωσης του ονόματος και τερματίζει στο κλείσιμο του μπλοκ που υποδηλώνεται με το σύμβολο }
 - Τοπικές μεταβλητές, παράμετροι συναρτήσεων
 - οι **static** μεταβλητές επίσης έχουν εμβέλεια μπλοκ
 - Άλλο το Storage class και άλλο η εμβέλεια
- **Εμβέλεια πρωτότυπου συνάρτησης**
 - Λίστα παραμέτρων
 - Τα ονόματα στο πρωτότυπο είναι προαιρετικά
 - Ο μεταγλωττιστής τα αγνοεί



Κανόνες Εμβέλειας

```
#include <iostream>
using std::cout;
using std::endl;

int x = 1;

int main () {
    cout << "global x is " << x << endl;
    int x = 3;
    cout << "local x in main is " << x << endl;
    cout << "global x is " << ::x << endl;
    {
        int x = 5;
        cout << "this block's x is" << x << endl;
    }
    return 0;
}
```

Καθολική εμβέλεια

Τοπική εμβέλεια μέσα στη συνάρτηση

Εμβέλεια κατά το χρόνο ζωής του μπλόκ.

Ο τελεστής :: μας επιτρέπει να πάρουμε την global μεταβλητή x αντί την τοπική με το ίδιο όνομα

```
global x is 1
local x in main is 3
this block's x is 5
```



Κανόνες Εμβέλειας

```
int useStaticLocal( ){  
    static int x = 5;    //  
    x+=5;  
}  
  
int main ( ){  
    for (int i=1; i<=5; i++)  
        cout << useStaticLocal() <<" ";  
}
```

Στατική τοπική μεταβλητή.
Αρχικοποιείται μόνο μια φορά και
διατηρεί την τιμή της μεταξύ των
κλήσεων της συνάρτησης

10 15 20 25 30



Αναδρομή

- Αναδρομικές συναρτήσεις
 - Συναρτήσεις που καλούν τον εαυτό τους
 - Λύνουν μόνον μια βασική περίπτωση
- Παράδειγμα - παραγοντικό

```
int factorial( int number ){  
    if ( number <= 1 )  
        return 1;  
    else  
        return number * factorial( number - 1 );  
}
```

```
int main () {  
    cout << factorial(5) <<" ";  
}
```



Inline Συναρτήσεις

- Inline functions
 - Χρειάζεται το Keyword **inline** πριν από τον ορισμό της συνάρτησης
 - Ζητά από τον μεταγλωττιστή να αντικαταστήσει κάθε κλήση της συνάρτησης στο πρόγραμμα μ' ένα αντίγραφο του κώδικα της συνάρτησης
 - Μειώνει την επιβάρυνση από την κλήση μιας συνάρτησης
 - Ο μεταγλωττιστής μπορεί να αγνοήσει την οδηγία **inline**
 - Καλή περίπτωση για μικρές και συχνά καλούμενες συναρτήσεις



Αναφορές Μεταβλητών

- Οι αναφορές μεταβλητών είναι συνώνυμα άλλων μεταβλητών
 - Ουσιαστικά πρόκειται για την ίδια μεταβλητή
 - Μπορεί να χρησιμοποιηθεί και εντός μιας συνάρτησης

```
int count = 1; // declare integer variable count
int &cRef = count; // create cRef as an alias for count
++cRef; // increment count (using its alias)
```

- Οι αναφορές πρέπει να αρχικοποιούνται όταν δηλώνονται
 - Διαφορετικά, ο μεταγλωττιστής παράγει error
 - Dangling reference
 - Αναφορά σε μη ορισμένη μεταβλητή



Υπερφόρτωση Συναρτήσεων (Function Overloading)

- Υπερφόρτωση Συναρτήσεων
 - Συναρτήσεις με το ίδιο όνομα και διαφορετικές παραμέτρους
 - Υλοποιούν παρόμοια δουλειά
 - Π.χ. συνάρτηση `square` για `ints` και συνάρτηση `square` για `floats`

```
int square( int x) {return x * x;}  
float square(float x) { return x * x; }
```
- Οι συναρτήσεις που είναι υπερφορτωμένες διαχωρίζονται από την υπογραφή τους
 - Με βάση το όνομα και τους τύπους των παραμέτρων (η σειρά παίζει ρόλο)
 - Η αλλαγή απλά στον τύπο του επιστρεφόμενου αποτελέσματος δεν είναι υπερφόρτωση!!! (αλλά λάθος μεταγλώττισης)



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 6** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 5: Η ΓΛΩΣΣΑ C++
Πίνακες & Δείκτες

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος
Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πίνακες

Πίνακες

- Τα στοιχεία ενός πίνακα συμπεριφέρονται ως μεταβλητές

- Ανάθεση/Εκτύπωση

```
c[ 0 ] = 3;
```

```
cout << c[ 0 ];
```

- Ο δείκτης ενός πίνακα μπορεί να προκύψει μετά από κάποιο υπολογισμό

```
c[ 5 - 2 ] ίδιο με c[3]
```



Παράδειγμα

Αρχικοποίηση, εκτύπωση πίνακα

```
int main() {  
    int n[ 10 ] = { 3, 6, 9, 12, 15 };  
    for ( int i = 0; i < 5; i++ )  
        cout << n[ i ] << endl;  
    return 0;  
}
```



Παράδειγμα

Αρχικοποίηση, εκτύπωση πίνακα

```
int main() {
    const int size=5;
    int n[size];
    for ( int i = 0; i < size; i++ )
        n[i] = i*5;
    for ( int i = 0; i < size; i++ )
        cout << n[i] <<" ";
    return 0;
}
```

Μόνο const επιτρέπεται (δεν αλλάζει) για το μέγεθος

0 5 10 15 20



Πίνακες Χαρακτήρων (Strings)

- Πίνακες χαρακτήρων
- Όλα τα strings τερματίζουν με το null χαρακτήρα (' \0')
- Παραδείγματα
 - `char string1[] = "hello";`
 - Το `Null` θα προστεθεί στο τέλος
 - Η μεταβλητή `string1` έχει 6 στοιχεία
 - `char string1[] = { 'h', 'e', 'l', 'l', 'o', '\0' };`
- Η προσπέλαση των στοιχείων γίνεται με τον ίδιο τρόπο
 - `string1[0]` is 'h'
 - `string1[2]` is 'l'



Πέρασμα πίνακα σε συνάρτηση

- Περνάμε το όνομα του πίνακα και προαιρετικά το μέγεθος του πίνακα
 - Παράδειγμα

```
int myArray[ 24 ];  
myFunction( myArray, 24 );
```
 - Το μέγεθος του πίνακα είναι καλή πρακτική να περνά ως όρισμα γιατί μας επιτρέπει να έχουμε loops προσπέλασης όλων των στοιχείων εντός της συνάρτησης
- Το πέρασμα του πίνακα γίνεται με αναφορά (by-reference)
 - Η συνάρτηση μπορεί να αλλάξει τις τιμές των στοιχείων του αρχικού πίνακα
 - Το όνομα του πίνακα αντιστοιχεί στη διεύθυνση του πρώτου στοιχείου του πίνακα

Πρωτότυπο συνάρτησης

```
void modifyArray( int b[], int arraySize );  
void modifyArray( int [], int );
```



Πολυδιάστατοι πίνακες

■ Δύο διαστάσεις

- `a[i][j]`
- Πίνακες με γραμμές και στήλες
- Πρώτα αναφέρουμε τη γραμμή και μετά τη στήλη
- “Array of arrays”
 - `a[0]` είναι ένας πίνακας 4 στοιχείων
 - `a[0][0]` είναι το πρώτο στοιχείο αυτού του πίνακα
- Αρχικοποίηση
 - Default τιμή 0
 - Η λίστα αρχικοποιεί κατά γραμμές

```
int a[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```



Πολυδιάστατοι πίνακες

```
void printArray( int [][] [ 3 ] );

int main(){
    int pinakas1[ 2 ] [ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
    int pinakas2[ 2 ] [ 3 ] = { 1, 2, 3, 4, 5 };
    int pinakas3[ 2 ] [ 3 ] = { { 1, 2 }, { 4 } };
    cout << "Pinakas1:" << endl;
    printArray( pinakas1 );
    cout << "Pinakas2:" << endl;
    printArray( pinakas2 ); cout << "Pinakas3:" << endl;
    printArray( pinakas3 );
    return 0;
}

void printArray( int a[][] [ 3 ] ) {
    for ( int i = 0; i < 2; i++ ) {
        for ( int j = 0; j < 3; j++ )
            cout << a[ i ] [ j ] << '.';
        cout << endl;
    }
}
```

```
Pinakas1:
1.2.3.
4.5.6.
Pinakas2:
1.2.3.
4.5.0.
Pinakas3:
1.2.0.
4.0.0.
```



Δείκτες

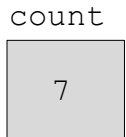
Δείκτες

- Δείκτες (Pointers)
 - Ισχυρό χαρακτηριστικό, αλλά δύσκολη η διαχείρισή τους
 - Υλοποιούν ένα είδος pass-by-reference
 - Στενή σχέση με τους πίνακες και τα strings



Δήλωση και αρχικοποίηση μεταβλητών τύπου δείκτη

- Μεταβλητές τύπου δείκτη
 - Αποθηκεύουν διευθύνσεις μνήμης ως τιμές
 - Μια κανονική μεταβλητή αποθηκεύει συγκεκριμένες τιμές (άμεση αναφορά - direct reference)
 - Ένας δείκτης αποθηκεύει τη διεύθυνση μιας μεταβλητής που περιέχει μια συγκεκριμένη τιμή (έμμεση αναφορά - indirect reference)



- Έμμεση αναφορά (Indirection)
 - Η αναφορά μιας τιμής μέσω ενός δείκτη



- Δήλωση δείκτη
 - Το `*` στη δήλωση υποδηλώνει ότι η μεταβλητή είναι δείκτης

```
int *myPtr;
```

- Οι δείκτες μπορούν να χρησιμοποιηθούν με οποιοδήποτε τύπο δεδομένων.
- Αρχικοποίηση δείκτη
 - Αρχικοποιείται στο **0** ή **NULL**



Τελεστές Δεικτών

- **&** (τελεστής διεύθυνσης)
 - Επιστρέφει **τη διεύθυνση** μνήμης του τελεστέου
 - Παράδειγμα

```
int y = 5;  
int *yPtr;  
yPtr = &y;
```



Τελεστές Δεικτών

- * (τελεστής έμμεσης αναφοράς ή αποαναφοράς)
 - Επιστρέφει **το αντικείμενο** που δείχνει ο δείκτης
 - ***yPtr** επιστρέφει το **y** (αφού **yPtr** δείχνει στο **y**).
 - Ένας αποαναφοροποιημένος δείκτης είναι ένα lvalue


```
*yPtr = 9; // assigns 9 to y
```
- * και & είναι αντίστροφοι τελεστές (αλληλοαναιρούνται)



Κλήση συναρτήσεων και πέρασμα ορισμάτων με αναφορά

- Pass-by-reference με ορίσματα δείκτες
 - Περνάμε τη διεύθυνση του ορίσματος χρησιμοποιώντας τον τελεστή διεύθυνσης &
 - Οι πίνακες δεν χρειάζονται τον τελεστή & αφού το όνομα του πίνακα αντιστοιχεί ήδη σε δείκτη
 - Χρησιμοποιούμε τον τελεστή έμμεσης αναφοράς * για την προσπέλαση των τιμών των μεταβλητών εντός της συνάρτησης



Παράδειγμα

```
void fooByValue( int x ) {
    x=x+5;
}
void fooByReference1( int* x ) {
    *x=*x+5;           // χρήση * για να πάρουμε την τιμή
}
void fooByReference2( int& x ) {
    x=x+5;
}
int main(){
    int val = 5;
    fooByValue(val );
    cout << val << endl;    // 5 δεν άλλαξε το val
    fooByReference1(&val);  //  χρήση & για να πάρουμε την διεύθυνση
    cout << val << endl;    // 10 - άλλαξε
    fooByReference2(val);
    cout << val << endl;    // 15 - άλλαξε
    return 0;
}
```

5
10
15



Χρήση του `const`

- **`const` qualifier**
 - Δεν επιτρέπει την αλλαγή της τιμής της μεταβλητής
 - Βάζουμε το **`const`** όταν η συνάρτηση δεν χρειάζεται να αλλάξει την τιμή της μεταβλητής
- Αρχή του ελαχίστου δικαιώματος
 - Μια συνάρτηση δεν χρειάζεται να αποκτά παραπάνω δικαιώματα από αυτά που χρειάζεται για την επίτευξη της εργασίας της
- **`const` pointers**
 - Πάντα δείχνουν στην ίδια θέση μνήμης
 - Η περίπτωση του ονόματος ενός πίνακα
 - Πρέπει να αρχικοποιείται όταν δηλώνεται



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαια 7, 8** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 6: C++ ΚΛΑΣΕΙΣ, ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ, ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Κλάσεις

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Τάξεις / Κλάσεις

Ορισμός Δομών-Structure

- Δομές
 - Καθορίζουν τύπους που προκύπτουν από τη συνάθροιση μελών άλλων τύπων

```
struct Time {  
    int hour;  
    int minute;  
    int second;  
};
```

- Ονοματολογία μελών σε δομή Structure
 - Στην ίδια **struct**: πρέπει να δίνονται μοναδικά ονόματα
 - Σε διαφορετικές **structs**: μπορούν τα ονόματα να διαμοιράζονται
- Ο ορισμός **struct** τελειώνει πάντα με ερωτηματικό



Ορισμός Δομών-Structure

- Αυτό-αναφερόμενη δομή-struct
 - Τα μέλη μιας Structure δε μπορεί να είναι εκφάνσεις (instance) εμφωλευμένης **struct**
 - Τα μέλη μιας Structure μπορούν να είναι δείκτης σε μια instance εμφωλευμένης **struct** (αυτό-αναφερόμενη structure)
 - Χρήσιμο για διασυνδεδεμένες λίστες, ουρές, στοίβες και δένδρα
- **struct**
 - Δημιουργεί έναν νέο τύπο δεδομένων που χρησιμοποιείται για να δηλώσει κανείς μεταβλητές
 - Δηλώνονται όπως και οι λοιπές μεταβλητές άλλων τύπων
 - Παραδείγματα:
 - `Time timeObject;`
 - `Time timeArray[10];`
 - `Time *timePtr;`
 - `Time &timeRef = timeObject;`



Προσπέλαση μελών Structure

- Τελεστές προσπέλασης μελών

- Τελεστής τελεία (.) για μέλη δομής-structure και τάξης
- Τελεστής βέλος (->) για μέλη δομής-structure και τάξης μέσω δείκτη σε αντικείμενο
- Εκτύπωση μέλους `hour` του `timeObject`:

```
cout << timeObject.hour;
```

Ή

```
timePtr = &timeObject;  
cout << timePtr->hour;
```

- `timePtr->hour` είναι όμοιο με (`*timePtr`).`hour`
 - Οι παρενθέσεις απαιτούνται
 - * έχει μικρότερη προτεραιότητα έναντι του .



Τάξη - Class

- Τάξεις
 - Αντικείμενα Model
 - Χαρακτηριστικά - Attributes (μέλη δεδομένα)
 - Συμπεριφορές - Behaviors (μέλη συναρτήσεις)
 - Ορίζεται με τη λέξη-κλειδί **class**
 - Μέλη συναρτήσεις
 - Μέθοδοι
 - Καλούνται σε απάντηση μηνύματος
- Καθοριστές πρόσβασης σε μέλη
 - **Δημόσια - public:**
 - Πρόσβαση σε οποιοδήποτε αντικείμενο της τάξης εντός περιοχής -scope
 - **Ιδιωτικό - private:**
 - Πρόσβαση μόνο από μέλη συναρτήσεις της τάξης
 - **Προστατευόμενο - protected:**
 - Πρόσβαση από παράγωγες τάξεις



Τάξη - Class

- Συνάρτηση Constructor
 - Ειδικό μέλος συνάρτηση
 - Αρχικοποιεί τα δεδομένα
 - Έχει το ίδιο όνομα με την τάξη
 - Καλείται όταν παράγεται το αντικείμενο
 - Μπορεί να υπάρχουν πολλοί constructors
 - Υπερφόρτωση συναρτήσεων Function
 - Υπάρχει default, καλύπτεται με ορισμό χωρίς arguments
 - Δεν έχει/επιστρέφει κάποιο type (εξ ορισμού void)
 - Public ή private



Παράδειγμα

```
class Time {  
    public: ←  
        Time();  
        void setTime( int, int, int );  
    private:  
        int hour;  
        int minute;  
        int second;  
};
```

Ομαδοποίηση των public / private μελών (δεν υποστηρίζεται στην JAVA)

```
void Time::setTime( int h, int m, int s ) {  
    hour = ( h >= 0 && h < 24 ) ? h : 0;  
    minute = ( m >= 0 && m < 60 ) ? m : 0;  
    Second = ( s >= 0 && s < 60 ) ? s : 0;  
}
```

Δυνατότητα υλοποίησης μεθόδων εκτός της κλάσης.

Χρήση τελεστή :: για να αναφερθούμε στην κλάση έξω από αυτήν.

```
int main() {  
    Time t;  
    t.setTime(2,20,45)  
}
```



Μέθοδος Καταστροφής (Destructor)

- Destructors

- Έχουν το ίδιο όνομα με την τάξη
 - Προηγείται το σημάδι μαθηματικής άρνησης (tilde) (~)
- Χωρίς παραμέτρους
- Δε μπορεί να υπερφορτωθεί
- Public



Πλεονεκτήματα

- Πλεονεκτήματα χρήσης τάξεων
 - Απλοποιούν τον προγραμματισμό
 - Διεπαφές
 - Κρύβουν την υλοποίηση
 - Επαναχρησιμοποίηση Κώδικα
 - Σύνθεση (συνάθροιση) - Composition (aggregation)
 - Αντικείμενα τάξης μπορούν να περιληφθούν ως μέλη άλλων τάξεων
 - Κληρονομικότητα
 - Νέες τάξεις προκύπτουν από παλαιές



Εμβέλεια τάξης και προσπέλαση μελών

- Εμβέλεια τάξης
 - Μέλη δεδομένα και συναρτήσεις
 - Εντός εμβέλειας
 - Μέλη τάξης
 - Άμεσα προσπελάσιμα από όλα τα μέλη συναρτήσεις
 - Αναφορά με το όνομα
 - Εκτός εμβέλειας τάξης
 - Αναφορά με handles
 - Όνομα αντικειμένου, αναφορά στο αντικείμενο, δείκτης στο αντικείμενο



Εμβέλεια τάξης και προσπέλαση μελών

■ Εμβέλεια συνάρτησης

- Μεταβλητές δηλώνονται σε συναρτήσεις μέλη
- Είναι γνωστές μόνο στη συνάρτηση
- Μεταβλητές με ίδιο όνομα με μεταβλητές εμβέλειας τάξης
 - Η εμβέλεια της μεταβλητής τάξης «κρύβεται»
 - Προσπέλαση με τελεστή καθορισμού εμβέλειας (: :)

ClassName :: classVariableName

- Οι μεταβλητές είναι γνωστές στις συναρτήσεις που ορίζονται
- Οι μεταβλητές καταστρέφονται μετά την ολοκλήρωση της συνάρτησης



Εμβέλεια τάξης και προσπέλαση μελών

- Τελεστές για προσπέλαση μελών τάξης
 - Ίδια με των **structs**
 - Επιλογή με τελεία (.)
 - Αντικείμενο
 - Αναφορά σε αντικείμενο
 - Επιλογή με βέλος (->)
 - Δείκτες



Παράδειγμα

```
Time t1;
```

```
Time *timePtr = &t1;
```

```
Time &timeRef = t1;
```

```
t1.hour = 8; ← Χρήση τελείας για την επιλογή μέλους  
δεδομένου από το αντικείμενο
```

```
timeRef.hour = 2; ← Χρήση τελείας για την επιλογή μέλους στην  
αναφορά timeRef στο αντικείμενο.
```

```
timePtr->hour = 3; ← Χρήση βέλους για το δείκτη timePtr στο  
αντικείμενο
```



Έλεγχος πρόσβασης στα μέλη

- Επίπεδα πρόσβασης
 - **private**
 - Default
 - Προσβάσιμο σε μέλη συναρτήσεις και **friends**
 - **public**
 - Προσβάσιμο σε κάθε συνάρτηση που χειρίζεται αντικείμενο της τάξης
 - **protected**



Μέθοδοι Get/Set

- Συνάρτηση Set
 - Πραγματοποιεί ελέγχους εγκυρότητας προτού γίνουν μετατροπές σε δεδομένα **private**
 - Ειδοποιεί αν υπάρχουν λανθασμένες τιμές
- Συνάρτηση Get
 - Ελέγχει τη μορφή των δεδομένων που επιστρέφονται



Προσοχή: Επιστροφή αναφοράς σε δεδομένα `private`

■ Αναφορά σε αντικείμενο

- Alias for name of object
- Lvalue
 - Μπορεί να λάβει τιμή σε ανάθεση
 - Αλλάζει το αρχικό αντικείμενο

■ Επιστροφή αναφοράς

- **public** συναρτήσεις μπορούν να επιστρέφουν μη-**const** αναφορές σε δεδομένα **private**
 - Κατά συνέπεια ο χρήστης μπορεί να τροποποιεί δεδομένα **private**



Default Ανάθεση

- Ανάθεση αντικειμένων
 - Τελεστής ανάθεσης (=)
 - Μπορεί να αναθέσει σε ένα αντικείμενο ένα άλλο ίδιου τύπου
 - Default ανάθεση
 - Κάθε δεξιό μέλος ανατίθεται ανεξάρτητα στο αριστερό μέλος
- Περνώντας και επιστρέφοντας αντικείμενα
 - Αντικείμενα περνούν ως παράμετροι συνάρτησης
 - Αντικείμενα επιστρέφονται από συναρτήσεις
 - Default πέρασμα με τιμή
 - Αντίγραφο του αντικείμενου που περνάει, επιστρέφεται
 - Αντίγραφο του constructor
 - Αντίγραφο των αρχικών τιμών σε νέο αντικείμενο



const (Σταθερά)

- Η αρχή της ελάχιστης πρόσβασης
 - Επιτρέπουμε πρόσβαση για τροποποιήσεις μόνο στα απαραίτητα αντικείμενα
- **const**
 - Ορίζει αντικείμενο που δε τροποποιείται
 - Δίνει Compiler error



Χρήση Const

- **const** μέθοδοι

- Οι μέθοδοι αντικειμένων **const** πρέπει να είναι και αυτές **const**

- Δε μπορεί να τροποποιούν αντικείμενα

- Ορίζουμε ως **const** σε

- Πρωτότυπο

- Μετά τη λίστα παραμέτρων

- Δηλώσεις

- Πριν την αρχή του αριστερού αγκίστρου



Αρχικοποίηση

- Αρχικοποίηση αντικειμένου
 - Αρχικοποίηση με `member initializer syntax`
 - Μπορεί να χρησιμοποιηθεί
 - Με όλα τα μέλη δεδομένων
 - Πρέπει να χρησιμοποιηθεί
 - Για τα μέλη `const`
 - Για όλες τις αναφορές μεταβλητών



Παράδειγμα

```
class myDate{
    public:
        myDate(int d, int m, int y) :day(d), month(m), year(y) { }
        ~myDate() {
            cout << "date object destroyed";
        }
        int getDay () const { // Δεν τροποποιεί το αντικείμενο
            return day;
        }
        void setDay (int d){
            day = (d>0 && d<32)? d : 1;
        }
    private:
        int day;
        int month;
        const int year;
};

int main() {
    myDate d1(2,11,2001);
    //cout << d1.day; Δεν μπορούμε γιατί είναι private μέλος
    cout << d1.getDay();
    const myDate d2(4,12,2011);
    //d2.setDay(3); Compiler error γιατί η μέθοδος δεν είναι const
    int day2 = d2.getDay();
    // Μπορούμε γιατί η μέθοδος έχει δηλωθεί const
}
```

αρχικοποίηση

Η const μεταβλητή year δεν μπορεί να τροποποιηθεί για κανένα αντικείμενο. Παίρνει τιμή μόνο κατά την αρχικοποίηση με τον παραπάνω τρόπο.

Το const αντικείμενο d2 δεν μπορεί να τροποποιηθεί. Εκτός του δημιουργού, μπορούμε να καλέσουμε μόνο const μεθόδους της κλάσης



Σύνθεση/ Composition:

Αντικείμενα ως μέλη τάξης

- Σύνθεση/ Composition
 - Μία τάξη έχει αντικείμενα άλλης τάξης ως μέλη
- Κατασκευή αντικειμένων
 - Τα μέλη αντικείμενα δημιουργούνται με τη σειρά που δηλώνονται
 - Δεν ακολουθείται η σειρά του constructor
 - Δημιουργούνται πριν από τα αντικείμενα της τάξης που τα χρησιμοποιεί



Παράδειγμα - Σύνθεση

```
class myEvent{
    public:
        myEvent(myDate str, myDate endd): startdate(str),
                                           enddate(endd) {};

        int getDuration() const;
    private:
        myDate startdate;
        myDate enddate;
};
```



friend Συναρτήσεις και Τάξεις

- **friend** συναρτήσεις
 - Ορίζονται εκτός εμβέλειας της τάξης
 - Έχουν πρόσβαση σε non-public members
- Δήλωση **friends**
 - Συνάρτηση
 - Προηγείται το keyword **friend**
 - Όλες οι συναρτήσεις της τάξης **classTwo** ως **friends** της τάξης **classOne**
 - Βάζουμε τη δήλωση της μορφής
`friend class classTwo;`
στον ορισμό της **classOne**



friend Συναρτήσεις και Τάξεις

- Ιδιότητες
 - Μπορεί να δοθεί όχι να ανακληθεί
 - τάξη **B friend** της τάξης **A**
 - Η τάξη **A** πρέπει να δηλώσει την τάξη **B** ως **friend**
 - Όχι συμμετρική
 - τάξη **B friend** της τάξης **A**
 - τάξη **A** όχι απαραίτητα **friend** της τάξης **B**
 - Όχι μεταβατική
 - τάξη **A friend** της **B**
 - τάξη **B friend** της **C**
 - τάξη **A** όχι απαραίτητα **friend** της **C**



Παράδειγμα

```
class myDate{  
    friend int yearDifference(myDate, myDate);  
public:  
    myDate(int d, int m, int y):day(d), month(m), year(y) {}  
private:  
    int day; int month; int year;  
};
```

```
int yearDifference(myDate dt1, myDate dt2){  
    return dt2.year - dt1.year;  
}
```

```
int main(){  
    myDate d1(2,11,2001);  
    myDate d2(3,4,2015);  
    cout << yearDifference(d1,d2);  
}
```

Η friend συνάρτηση δεν είναι μέλος της κλάσης.

- Δεν χρειάζεται ο τελεστής :: στην δήλωση
- Δεν καλείται από αντικείμενα της κλάσης

Απευθείας πρόσβαση στα private μέλη της friend κλάσης.



Χρήση του `this`

■ `this`

- Επιτρέπει στο αντικείμενο να έχει πρόσβαση στη δική του διεύθυνση
- Ο τύπος του δείκτη `this` εξαρτάται από:
 - Τύπο του αντικειμένου
 - Αν η συνάρτηση είναι `const`
 - Για τις `non-const` συναρτήσεις `Employee`
 - `this` έχει τύπο `Employee * const`
Constant δείκτη σε `non-const Employee` αντικείμενο
 - Για τις `const` συναρτήσεις `Employee`
 - `this` έχει τύπο `const Employee * const`
Constant δείκτη σε `constant Employee` αντικείμενο



Χρήση του `this`

- Σειριακή κλήση συναρτήσεων

- Πολλαπλές συναρτήσεις καλούνται με μία δήλωση
- Η συνάρτηση επιστρέφει δείκτη αναφοράς στο ίδιο το αντικείμενο

```
{ return *this; }
```

- Οι συναρτήσεις που δεν επιστρέφουν αναφορές πρέπει να κληθούν τελευταίες



Διαχείριση Δυναμικής Μνήμης

- Διαχείριση δυναμικής μνήμης
 - Ελέγχει τη διανομή μνήμης
 - Με χρήση των τελεστών **new** και **delete**
 - include standard header **<new>**



Διαχείριση Δυναμικής Μνήμης

- Έστω

```
Time *timePtr;  
timePtr = new Time;
```

- Τελεστής **new**

- Δημιουργεί αντικείμενα κατάλληλου μεγέθους για τον τύπο **Time**
 - Δίνει λάθος αν δεν υπάρχει χώρος στη μνήμη
- Επιστρέφει δείκτη στον συγκεκριμένο τύπο

- Με αρχικοποίηση

```
double *ptr = new double( 3.14159 );  
Time *timePtr = new Time( 12, 0, 0 );
```

- Δήλωση πίνακα

```
int *gradesArray = new int[ 10 ];
```



Διαχείριση Δυναμικής Μνήμης

- Έστω

```
delete timePtr;
```

- Τελεστής **delete**

- Καλεί το destructor
- Η μνήμη μπορεί να χρησιμοποιηθεί με άλλα αντικείμενα

- Deallocating arrays

```
delete [] gradesArray;
```

- Απελευθερώνει το array στο οποίο δείχνει το **gradesArray**
- Αν είναι δείκτης σε array αντικειμένων
 - Καλείται πρώτα ο destructor για κάθε αντικείμενο του array
 - Μετά απελευθερώνει τη μνήμη



Μεταβλητές Τάξης (Static)

- **static** μεταβλητή τάξης
 - Δεδομένα διαθέσιμα σε όλη την τάξη
 - Ιδιότητα της τάξης, όχι συγκεκριμένου αντικειμένου της τάξης
 - Αποδοτικό όταν απλά ένα αντίγραφο της τάξης είναι αρκετό
 - Μόνο η μεταβλητή **static** πρέπει να ενημερώνεται
 - Μπορεί να μοιάζει με `global`, αλλά έχει εμβέλεια στην τάξη
 - Αρχικοποιείται μια μόνο φορά
 - Υπάρχει ακόμη και χωρίς αντικείμενο



Παράδειγμα

```
class myDate{
public:
    myDate(int d, int m, int y) :day(d), month(m), year(y) {
        if (y>maximumYear)
            maximumYear=y;
    }
    int static getMaxYear(){ return maximumYear ;}

private:
    int day; int month; int year;
    static int maximumYear;
};
```

Static μέθοδος που επιστρέφει την static μεταβλητή

```
int myDate::maximumYear=0;      Αρχικοποίηση έξω από την κλάση.
```

```
int main() {
    cout<< myDate::getMaxYear()<<endl;
    myDate d1(2,11,2001); myDate d2(3,4,2015);
    cout<< myDate::getMaxYear();
}
```

Κλήση και χωρίς να υπάρχει ακόμα αντικείμενο.



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαια 3, 9** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
[http://media.pearsoncmg.com/ph/esm/deitel/cpp htp 9/code examples/Code Examples.zip](http://media.pearsoncmg.com/ph/esm/deitel/cpp_htp_9/code_examples/Code_Examples.zip)



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 6: C++ ΚΛΑΣΕΙΣ, ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ, ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Κληρονομικότητα

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Κληρονομικότητα

Κληρονομικότητα

- Η κληρονομικότητα αφορά
 - Επαναχρησιμοποίηση λογισμικού
 - Δημιουργία νέας κλάσης (**παραγόμενη κλάση**) από μια αρχική υπάρχουσα κλάση (**κλάση βάσης**)
 - Κληρονομούνται τα χαρακτηριστικά και η συμπεριφορά της αρχικής κλάσης
 - Επέκταση της παραγόμενης κλάσης με νέες δυνατότητες (customization)
 - νέα πεδία
 - επιπρόσθετη συμπεριφορά



Τύποι Κληρονομικότητας

- Υποστηρίζονται 3 τύποι κληρονομικότητας
 - **public**
 - Κάθε αντικείμενο μιας παραγόμενης κλάσης είναι αντικείμενο και της κλάσης βάσης
 - Αντικείμενα μιας κλάσης βάσης δεν είναι αντικείμενα της παραγόμενης κλάσης
 - Παράδειγμα: Όλα τα αυτοκίνητα είναι οχήματα, αλλά δεν ισχύει το αντίστροφο
 - Επιτρέπεται η προσπέλαση των μη-**private** μελών της κλάσης βάσης
 - Η παραγόμενη κλάση μπορεί να επιφέρει αλλαγές στα **private** μέλη της κλάσης βάσης, μέσω κληρονομούμενων μη-**private** μεθόδων
 - **private**
 - Αντίστοιχη με τη σχέση σύνθεσης
 - **protected**
 - Χρησιμοποιείται σπάνια

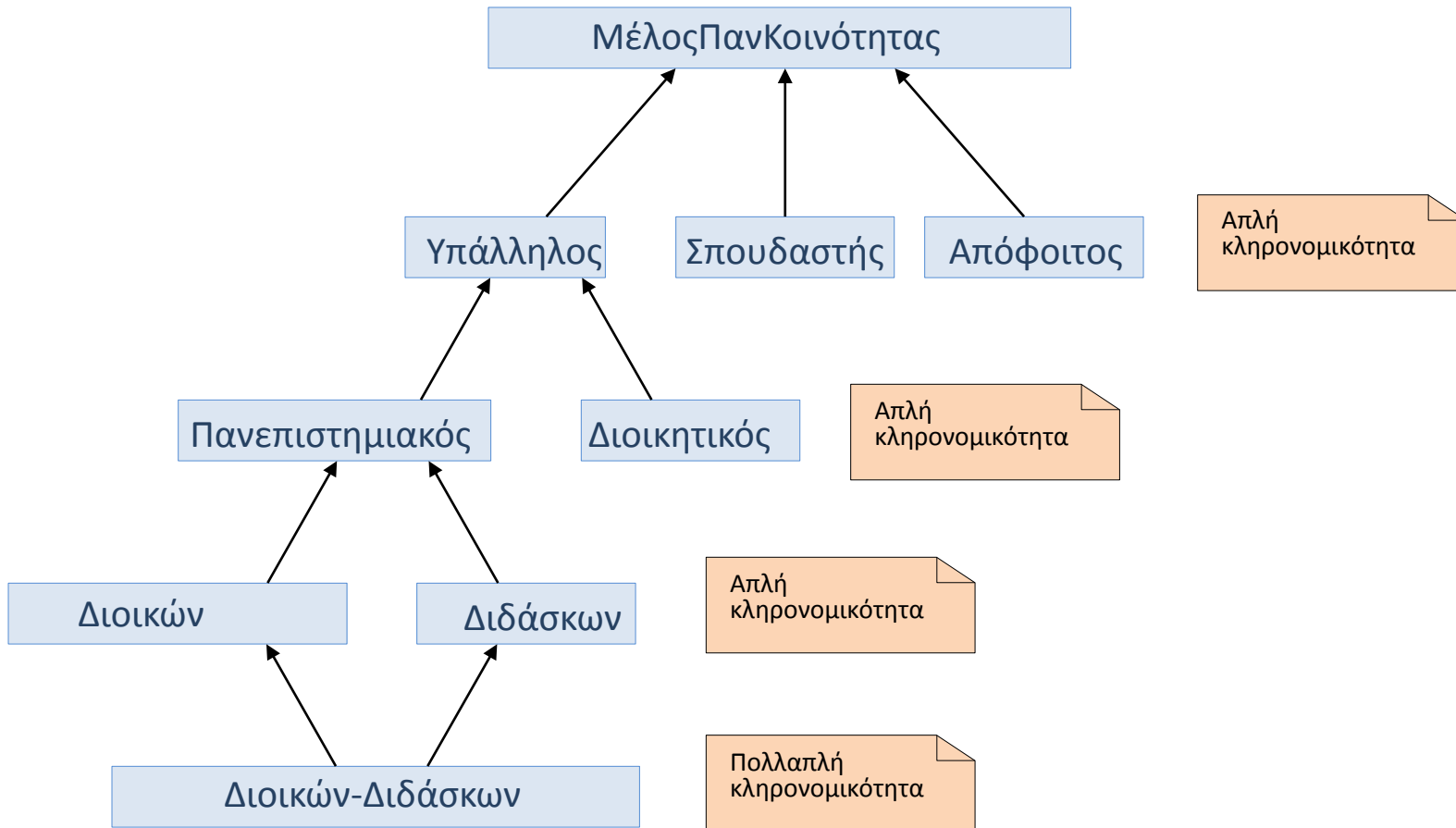


Ιεραρχία κλάσεων

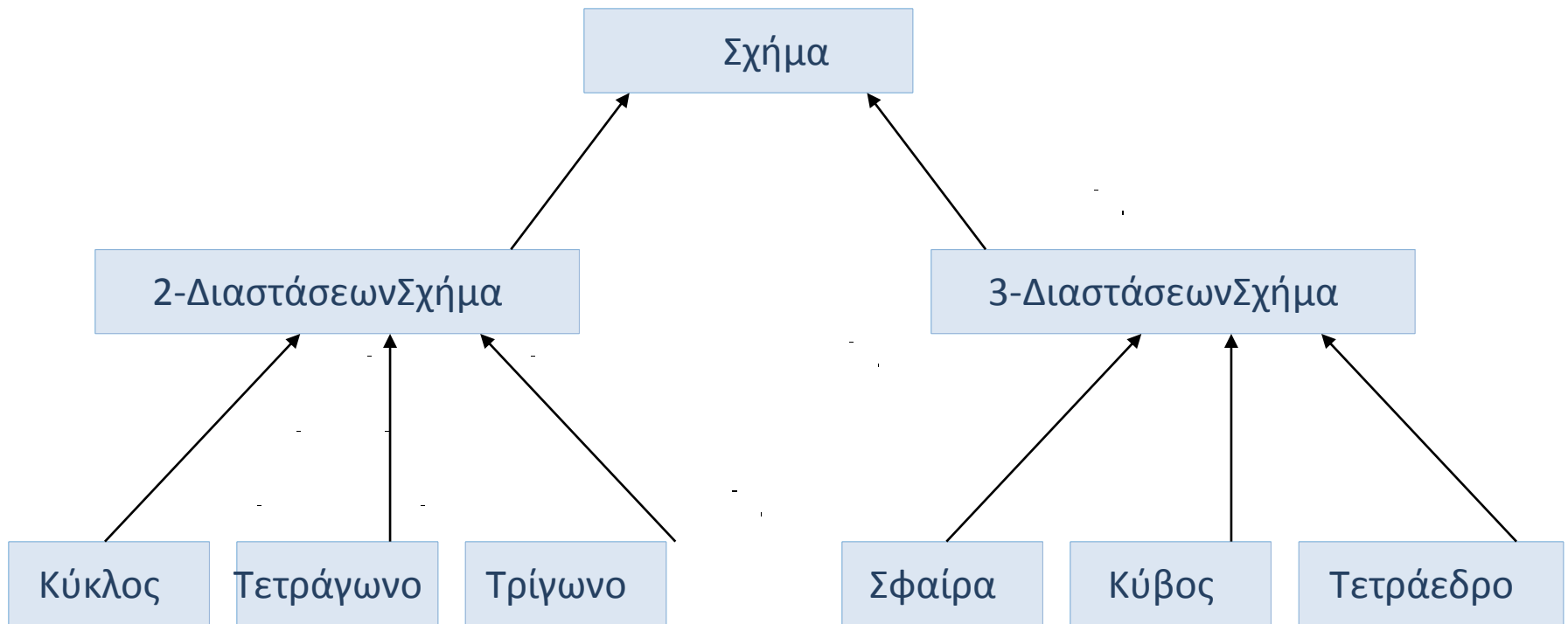
- Άμεση κλάση βάσης
 - Κληρονομείται απ' ευθείας (ιεραρχία ενός επιπέδου)
- Έμμεση κλάση βάσης
 - Κληρονομείται σε ιεραρχία δύο ή περισσότερων επιπέδων
- Απλή κληρονομικότητα
 - Μία παραγόμενη κλάση συνδέεται με μία μόνο κλάση βάσης
- Πολλαπλή κληρονομικότητα
 - Μία παραγόμενη κλάση συνδέεται με πολλές κλάσεις βάσης
 - Να χρησιμοποιείται με προσοχή



Παράδειγμα Ιεραρχίας



Παράδειγμα Ιεραρχίας



Κληρονομικότητα Public

- **public** τύπος κληρονομικότητας

- Ορίζεται με την εντολή:

```
Class TwoDimensionalShape : public Shape
```

- Προσπέλαση των **private** μελών της κλάσης βάσης

- Δεν μπορεί να γίνει απ' ευθείας
- Παρόλα αυτά τα private μέλη κληρονομούνται και μπορούμε να τα χειριστούμε μέσω των μη-private κληρονομούμενων μεθόδων

- Προσπέλαση των **public** και **protected** μελών της κλάσης βάσης

- Κληρονομούνται και είναι δυνατή η απ' ευθείας προσπέλασή τους (με χρήση του ονόματος του μέλους)

- **friend** functions

- Δεν κληρονομούνται



Protected Μέλη

- **protected** προσπέλαση
 - Ενδιάμεσο επίπεδο προστασίας δεδομένων μεταξύ **public** και **private**
 - Η προσπέλαση των **protected** μελών είναι εφικτή σε:
 - μέλη της κλάσης βάσης
 - **friends** της κλάσης βάσης
 - μέλη της παραγόμενης κλάσης
 - **friends** της παραγόμενης κλάσης



Παράδειγμα

```
class myDate{
    public: myDate(int d, int m, int y) :day(d), month(m), year(y) {}
    void print(){
        cout << year <<"/"<<month<<"/"<<day;
    }
    protected:
        int day, month, year;
};
```

Η myDateTime κληρονομεί την myDate

```
class myDateTime : public myDate{
    public:
        myDateTime(int d, int m, int y, int se, int mi, int ho)
            : myDate(d,m,y), sec(se), minute(mi), hour(ho) {}
        void print(){
            myDate::print();
            cout <<" - " << hour <<":" <<minute<<":" <<sec;
        }
        void incrementDay(){
            day++;
        }
    private:
        int sec, minute, hour;
};
```

Δεν υπάρχει super όπως στην Java

- Κλήση του Δημιουργού της myDate
- Κλήση της print της myDate

Άμεση πρόσβαση στα protected μέλη της κλάσης myDate



Χρήση Protected Μελών

- Χρήση **protected** μελών

- + **Πλεονεκτήματα**

- Οι παραγόμενες κλάσεις μπορούν να αλλάξουν τις τιμές των πεδίων απ' ευθείας
 - Μικρή βελτίωση της ταχύτητας
 - Αποφεύγουμε την κλήση των μεθόδων set/get

- **Μειονεκτήματα**

- Δεν προσφέρεται για έλεγχο εγκυρότητας τιμών
 - Η παραγόμενη κλάση μπορεί να δώσει μη-επιτρεπτή τιμή
 - Δημιουργία σχέσεων εξάρτησης
 - Οι μέθοδοι της παραγόμενης κλάσης είναι πιο πιθανόν τώρα να εξαρτώνται από την υλοποίηση της κλάσης βάσης
 - Εάν αλλάξει η υλοποίηση της κλάσης βάσης μπορεί να χρειαστεί να τροποποιήσουμε και την παραγόμενη κλάση
 - » Εύθραυστο λογισμικό



Δημιουργία αντικειμένου

- Δημιουργία αντικειμένου παραγόμενης κλάσης
 - Σειρά κλήσης των constructors
 - Ο constructor της παραγόμενης κλάσης καλεί τον constructor της κλάσης βάσης
 - Έμμεσα ή άμεσα
 - Βάση της ιεραρχίας κληρονομικότητας
 - Ο τελευταίος στη σειρά constructor που καλείται και ο πρώτος που ολοκληρώνει την εκτέλεσή του
 - Αρχικοποίηση των πεδίων
 - Ο constructor κάθε κλάσης βάσης αρχικοποιεί τα πεδία του



Καταστροφή αντικειμένου

- Καταστροφή αντικειμένου παραγόμενης κλάσης
 - Σειρά κλήσης των destructors
 - Αντίστροφη σειρά από αυτή της κλήσης των constructors
 - Ο destructor της παραγόμενης κλάσης καλείται πρώτος
 - Ο destructor της επόμενης κλάσης βάσης στην ιεραρχία καλείται στη συνέχεια
 - Συνεχίζουμε προς τα πάνω μέχρι να φθάσουμε στην τελευταία κλάση βάσης της ιεραρχίας
 - Μετά την εκτέλεση του τελευταίου destructor, το αντικείμενο αφαιρείται από τη μνήμη



Κληρονομικότητα

- Constructors, destructors, τελεστές ανάθεσης της κλάσης βάσης
 - Δεν κληρονομούνται από τις παραγόμενες κλάσεις
 - Οι constructors, τελεστές ανάθεσης της παραγόμενης κλάσης μπορούν να καλέσουν
 - Constructors
 - Τελεστές ανάθεσης



Κληρονομικότητα

public, protected και private

Προσδιοριστής προσπέλασης μέλους κλάσης βάσης	Τύπος κληρονομικότητας		
	public κληρονομικότητα	protected κληρονομικότητα	private κληρονομικότητα
Public	public στην παραγόμενη κλάση. Μπορεί να προσπελασθεί απ' ευθείας από όλες τις μη- static μεθόδους, friend και εξωτερικές συναρτήσεις.	protected στην παραγόμενη κλάση. Μπορεί να προσπελασθεί απ' ευθείας από όλες τις μη- static μεθόδους και friend συναρτήσεις.	private στην παραγόμενη κλάση. Μπορεί να προσπελασθεί απ' ευθείας από όλες τις μη- static μεθόδους και friend συναρτήσεις.
Protected	protected στην παραγόμενη κλάση. Μπορεί να προσπελασθεί απ' ευθείας από όλες τις μη- static μεθόδους και friend συναρτήσεις.	protected στην παραγόμενη κλάση. Μπορεί να προσπελασθεί απ' ευθείας από όλες τις μη- static μεθόδους και friend συναρτήσεις.	private στην παραγόμενη κλάση. Μπορεί να προσπελασθεί απ' ευθείας από όλες τις μη- static μεθόδους και friend συναρτήσεις.
Private	Κρυφή στην παραγόμενη κλάση. Μπορεί να προσπελασθεί από μη- static μεθόδους και friend συναρτήσεις μέσω public ή protected μεθόδους της κλάσης βάσης.	Κρυφή στην παραγόμενη κλάση. Μπορεί να προσπελασθεί από μη- static μεθόδους και friend συναρτήσεις μέσω public ή protected μεθόδους της κλάσης βάσης.	Κρυφή στην παραγόμενη κλάση. Μπορεί να προσπελασθεί από μη- static μεθόδους και friend συναρτήσεις μέσω public ή protected μεθόδους της κλάσης βάσης.



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαιο 11** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 6: C++ ΚΛΑΣΕΙΣ, ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ, ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Πολυμορφισμός

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πολυμορφισμός

Εισαγωγή

- Πολυμορφισμός
 - Απαιτείται η ύπαρξη μιας ιεραρχίας κλάσεων
 - Χρησιμοποιούμε αντικείμενα που ανήκουν στην ίδια ιεραρχία κλάσεων ως να ήταν όλα αντικείμενα της κλάσης βάσης
 - Εικονικές συναρτήσεις + δυναμική σύνδεση (Virtual functions + dynamic binding)
 - Τεχνολογίες που υλοποιούν τον πολυμορφισμό
 - Επιτρέπει την αποτελεσματική επέκταση των προγραμμάτων
 - Νέες κλάσεις μπορούν εύκολα να προστεθούν και να υποστούν επεξεργασία όπως και οι υπάρχουσες



Παράδειγμα

- Κλάση βάσης: *Animal*
- Παραγόμενες κλάσεις: *Fish, Frog, Bird*
- Η μέθοδος *move()* είναι κοινή, αλλά υλοποιείται διαφορετικά για κάθε είδος *Animal*.
- Μια εφαρμογή κρατά έναν πίνακα από αντικείμενα των παραγόμενων κλάσεων και κάθε 1sec στέλνει ένα μήνυμα *move* σε κάθε αντικείμενο.
- Κάθε αντικείμενο τύπου *Animal* που δέχεται το μήνυμα γνωρίζει πως θα πρέπει να αντιδράσει.
- Το ίδιο μήνυμα που στέλνεται σε διαφορετικά αντικείμενα (του ίδιου βασικού τύπου *Animal*) έχει πολλές μορφές αποτελεσμάτων → πολυμορφισμός
- Πόσο εύκολο είναι να προσθέσουμε μια νέα κλάση *Turtle*?



Παράδειγμα

```
class Animal{
public:
    Animal(char* n):name(n){}
    void print() const{
        cout << "-----\n";
        cout << name << " is an animal";
    }
private:
    char* name;
};
class Dog : public Animal{
public:
    Dog(char* n):Animal(n){}
    void print() const{
        Animal::print();
        cout << ", and a dog." << endl; }
};
class Cat : public Animal{
public:
    Cat(char* n):Animal(n){}
    void print() const{
        Animal::print();
        cout << ", and a cat." << endl;
    }
};
class Bird : public Animal{
public:
    Bird(char* n):Animal(n){}
    void print() const{
        Animal::print();
        cout << ", and a bird." << endl;
    }
};
```

Υπερκάλυψη της μεθόδου print στις παράγωγες κλάσεις

```
int main() {
    Dog d("Pluto");
    d.print();
    Cat c("Tom");
    c.print();
    Bird b("Twitty");
    b.print();
```

```
    Animal *animalPointer = &d;
    animalPointer->print();
}
```

Ανάθεση αντικειμένου παράγωγης κλάσης σε δείκτη της κλάσης βάσης.

```
-----
Pluto is an animal, and a dog.
-----
Tom is an animal, and a cat.
-----
Twitty is an animal, and a bird.
-----
Pluto is an animal
```

Η μέθοδος που θα κληθεί εξαρτάται από την κλάση του δείκτη, (εδώ είναι τύπου Animal)

Διαφορετική συμπεριφορά από την Java



Παράδειγμα - virtual

```
class Animal{
public:
    Animal(char* n):name(n){}
    virtual void print() const{
        cout << "-----\n";
        cout << name << " is an animal";
    }
private:
    char* name;
};

class Dog : public Animal{
public:
    Dog(char* n):Animal(n){}
    void print() const{
        Animal::print();
        cout << ", and a dog." << endl; }
};

class Cat : public Animal{
public:
    Cat(char* n):Animal(n){}
    void print() const{
        Animal::print();
        cout << ", and a cat." << endl;
    }
};

class Bird : public Animal{
public:
    Bird(char* n):Animal(n){}
    void print() const{
        Animal::print();
        cout << ", and a bird." << endl;
    }
};
```

Αλλαγή της προκαθορισμένης συμπεριφοράς με virtual δήλωση

```
int main() {
    Dog d("Pluto");
    d.print();
    Cat c("Tom");
    c.print();
    Bird b("Twitty");
    b.print();

    Animal *animalPointer = &d;
    animalPointer->print();
}
```

```
-----
Pluto is an animal, and a dog.
-----
Tom is an animal, and a cat.
-----
Twitty is an animal, and a bird.
-----
Pluto is an animal, and a dog.
```

Η μέθοδος που θα κληθεί εξαρτάται από την πραγματική κλάση του αντικειμένου (δημιουργήθηκε ως Dog)



Δείκτης παραγόμενης κλάσης σε αντικείμενο της κλάσης βάσης

- Στο προηγούμενο παράδειγμα είδαμε
 - Δείκτης κλάσης βάσης σε αντικείμενο της παραγόμενης κλάσης
 - **Dog is an Animal**
- Δείκτης παραγόμενης κλάσης σε αντικείμενο της κλάσης βάσης
 - Compiler error
 - **Animal** is not a **Dog**
 - Η κλάση **Dog** μπορεί να διαθέτει ιδιότητες/λειτουργίες που δεν διαθέτει η κλάση **Animal**
 - Μπορούμε να κάνουμε cast τη διεύθυνση του αντικειμένου της κλάσης βάσης σε δείκτη παραγόμενης κλάσης
 - Ονομάζεται downcasting
 - Επιτρέπει την προσπέλαση λειτουργικότητας της παραγόμενης κλάσης

```
int main() {  
    Animal a("Unknown");  
    Dog d("Pluto");  
  
    Animal* animalPointer = &d;  
    Dog* dogPointer = &a;  
}
```



Κλήση μεθόδων παραγόμενης κλάσης μέσω δείκτη κλάσης βάσης

- Χρήση δείκτη/αναφοράς
 - Δείκτης κλάσης βάσης μπορεί να δείχνει σε αντικείμενο της παραγομένης κλάσης
 - Μπορεί όμως να καλέσει μόνο τις μεθόδους της κλάσης βάσης
 - Η κλήση μεθόδων της παραγόμενης κλάσης συνιστά λάθος
 - Μέθοδοι που δεν ορίζονται στη κλάση βάσης
- Ο τύπος δεδομένων ενός/μιας δείκτη/αναφοράς προσδιορίζουν τι μεθόδους μπορούμε να καλέσουμε



Εικονικές Συναρτήσεις (Virtual Functions)

- Ο κανόνας είναι ο τύπος του δείκτη να καθορίζει τι μέθοδοι μπορούν να κληθούν
- Οι **virtual** functions μπορούν να τον αλλάξουν
 - το αντικείμενο (και όχι ο δείκτης) να καθορίζει τι μέθοδοι μπορούν να κληθούν



Εικονικές Συναρτήσεις (Virtual Functions)

- Δηλώνουμε την `print` ως `virtual` στην κλάση βάσης
 - Υπερκαλύπτουμε τη `print` σε κάθε παραγόμενη κλάση
 - σαν να την ξαναορίζουμε, αλλά ή νέα συνάρτηση πρέπει να έχει ακριβώς την ίδια υπογραφή με αυτή της κλάσης βάσης
 - Εάν δηλώσουμε μια συνάρτηση `virtual` στην κλάση βάσης
 - `virtual void print() const;`
 - είναι `virtual` σε όλες τις παραγόμενες κλάσεις
 - Είναι καλή πρακτική να δηλώνουμε ρητά ως `virtual` τις συναρτήσεις και στις παραγόμενες κλάσεις (program clarity)
- Δυναμική σύνδεση (Dynamic binding)
 - Επιλογή κλήσης της κατάλληλης συνάρτησης κατά το χρόνο εκτέλεσης
 - Συμβαίνει μόνο με τη χρήση δεικτών/αναφορών
 - Εάν η συνάρτηση κληθεί με χρήση ονόματος αντικειμένου (π.χ. `dogObject.print()`), τότε χρησιμοποιείται η συνάρτηση του αντικειμένου (static binding)



Χρήση πεδίου για τύπο αντικειμένων και εντολής `switch`

- Ένας τρόπος να προσδιορίσουμε την κλάση ενός αντικειμένου
 - Βάζουμε στην κλάση βάσης ένα πεδίο
 - `shapeType` στην κλάση `Shape`
 - Χρησιμοποιούμε μια `switch` για να καλέσουμε τη σωστή `print` συνάρτηση
- Πολλά προβλήματα
 - Μπορεί να ξεχάσουμε τον έλεγχο κάποιας περίπτωσης στην `switch`
 - Εάν προσθέσουμε/αφαιρέσουμε μια κλάση πρέπει να ενημερώσουμε την `switch`
 - Χρονοβόρο και επιρρεπές σε λάθη
- Καλύτερα με πολυμορφισμό
 - απλούστερα προγράμματα, λιγότερο debugging



Abstract Classes

- Abstract class
 - Σκοπός: να αποτελέσει μια base class (abstract base class)
 - Ελλιπής
 - Συμπληρώνεται με τις παραγόμενες κλάσεις
 - Δεν κατασκευάζουμε αντικείμενα από μια abstract class
 - Μπορούμε να έχουμε δείκτες και αναφορές
- Concrete classes
 - Μπορούμε να κατασκευάσουμε αντικείμενα
 - Υλοποιούμε όλες τις συναρτήσεις που περιέχουν



Abstract Classes

- Είναι χρήσιμες, όχι υποχρεωτικές
- Για να ορίσουμε μια abstract class
 - Δεν υπάρχει το keyword abstract όπως στην Java
 - Θέλουμε μια ή περισσότερες "pure" virtual functions

```
virtual void print() const = 0;
```
 - Regular virtual functions
 - Έχουν υλοποίηση, η υπερκάλυψη είναι προαιρετική
 - Pure virtual functions
 - Δεν έχουν υλοποίηση, η υπερκάλυψη είναι υποχρεωτική
 - Μια abstract class μπορεί να έχει δεδομένα και υλοποιημένες συναρτήσεις



Παράδειγμα

```
#include <vector>
using std::vector;
...

void virtualViaReference( const Animal &baseClassRef ){
    baseClassRef.print();
}

void virtualViaPointer( const Animal *baseClassPtr ){
    baseClassPtr->print();
}

int main() {
    Animal a("Unknown");
    Dog d("Pluto");
    Cat c("Tom");
    Bird b("Twitty");
    vector< Animal* > animalVector( 3 );
    animalVector [ 0 ] = &d;
    animalVector [ 1 ] = &c;
    animalVector [ 2 ] = &b;
    cout << " VIRTUAL VIA REFERENCE" <<endl;
    for ( int i = 0; i < animalVector.size(); i++ )
        virtualViaReference( *animalVector[ i ] );
    cout <<endl<<endl<< " VIRTUAL VIA POINTER" <<endl;
    for ( int i = 0; i < animalVector.size(); i++ )
        virtualViaPointer( animalVector[ i ] );
}
```



Virtual Destructors

- Δείκτης κλάσης βάσης σε αντικείμενο της παραγόμενης κλάσης
 - Εάν καταστρέψουμε το αντικείμενο με την **delete**, η συμπεριφορά είναι απροσδιόριστη
- Απλή λύση
 - Δηλώνουμε τον destructor της base-class ως virtual
 - Όταν καλείται η **delete** καλείται και ο κατάλληλος destructor
- Όταν καταστρέφουμε ένα αντικείμενο μιας παραγόμενης κλάσης
 - Πρώτα εκτελείται ο destructor της παραγόμενης κλάσης
 - Μετά εκτελείται ο destructor της base-class
- Οι constructors δεν μπορεί να είναι virtual



Παράδειγμα

```
#include <vector>
#include <typeinfo>
using std::vector;
...
```

```
int main() {
    vector< Animal* > animalVector( 3 );
    animalVector [ 0 ] = new Dog("Pluto");
    animalVector [ 1 ] = new Cat("Tom");
    animalVector [ 2 ] = new Bird("Twitty");

    for ( int i = 0; i < animalVector.size(); i++ ){
        Dog *dogPtr = dynamic_cast<Dog *>( animalVector[ i ] );
        if (dogPtr) animalVector[i]->print();
    }

    for ( int i = 0; i < animalVector.size(); i++ ){
        cout <<endl<< "deleting object " << typeid (*animalVector[ i ] ).name() ;
        delete animalVector[i];
    }
}
```

Η `dynamic_cast` επιστρέφει null αν δεν μπορεί να γίνει το casting

Η `typeid` επιστρέφει αντικείμενο `type_info`, το οποίο περιέχει πληροφορίες όπως το όνομα του αντικειμένου με την μέθοδο `name()`

Pluto is an animal, and a dog.

deleting object 3Dog
deleting object 3Cat
deleting object 4Bird



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαιο 12** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 7: C++ TEMPLATES, ΥΠΕΡΦΟΡΤΩΣΗ ΤΕΛΕΣΤΩΝ, ΕΞΑΙΡΕΣΕΙΣ

Templates

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Templates

Εισαγωγή

- Templates
 - Templates Συναρτήσεων
 - Ορισμός μιας σειράς σχετιζόμενων συναρτήσεων (με υπερφόρτωση)
 - Templates Τάξεων
 - Ορισμός μιας σειράς σχετικών κλάσεων



Function Templates

- Υπερφορτωμένες Συναρτήσεις (Overloaded)
 - Παρόμοιες λειτουργίες
 - Διαφορετικός τύπος δεδομένων
- Πρότυπες Συναρτήσεις (templates)
 - Ίδια ακριβώς λειτουργία
 - Διαφορετικός τύπος δεδομένων
 - Δήλωση μιας μόνο συνάρτησης template
 - Ο compiler παράγει ξεχωριστές συναρτήσεις
 - Type checking



Function Templates

- Ορισμός Function template
 - Ορισμός με την λέξη κλειδί **template**
 - Ο τύπος των παραμέτρων δηλώνεται μέσα σε brackets **< >**
 - Πριν από κάθε παράμετρο μπαίνει το: **class** ή **typename** (ισοδύναμα)

```
template< class T >
```

```
template< typename ElementType >
```

```
template< class BorderType, class FillType >
```
 - Μπορούμε να καθορίσουμε τον τύπο σε:
 - Ορίσματα συνάρτησης
 - Τύπος επιστρεφόμενης τιμής
 - Τοπικές Μεταβλητές μέσα στο σώμα της συνάρτησης



Παράδειγμα

```
#include <iostream>
using namespace std;
```

```
template <class T>
class mypair {
    T a, b;
    public: mypair (T first, T second) {
        a=first;
        b=second;
    } T getmax ();
};
```

```
template <class T> T mypair<T>::getmax () {
    T retval;
    retval = a>b? a : b;
    return retval;
}
```

```
int main () {
    mypair <int> integerpair (100, 75);
    cout << integerpair.getmax();
    mypair <char> charpair ('g', 'c');
    cout << charpair.getmax();
    return 0;
}
```

Η κλάση mypair περιέχει δύο όμοιες μεταβλητές. Ο τύπος των μεταβλητών θα καθορίζεται κάθε φορά κατά την δημιουργία ενός αντικειμένου.

Ο τύπος μπορεί να είναι build in όπως στο παράδειγμα (int, char) αλλά και οποιασδήποτε κλάσης.

ΠΧ:

```
Dog dog1("Max")
```

```
Dog dog2("Wolfy");
```

```
mypair <Dog> charpair (dog1, dog2);
```

Προσοχή: στο συγκεκριμένο παράδειγμα θα πρέπει στην κλάση Dog να έχουμε κάνει υπερφόρτωση του τελεστή σύγκρισης >



Υλοποίηση Στοίβας με Template Κλάση

```
template< class T >
class Stack {
public:
    Stack( int = 10 );
    ~Stack() {
        delete [] stackPtr;
    }
    bool push( const T& );
    bool pop( T& );
    bool isEmpty() const { return top == -1; }
    bool isFull() const { return (top == size - 1); }
}

private:
    int size;
    int top;
    T *stackPtr; };
```

```
template< class T >
Stack< T >::Stack( int s ){
    size = s > 0 ? s : 10;
    top = -1;
    stackPtr = new T[ size ];
}
```

```
template< class T >
bool Stack< T >::push( const T &pushValue ){
    if ( !isFull() ) {
        stackPtr[ ++top ] = pushValue;
        return true;
    }
    return false;
}
```

```
template< class T >
bool Stack< T >::pop( T &popValue ){
    if ( !isEmpty() ) {
        popValue = stackPtr[ top-- ];
        return true;
    }
    return false;
}
```



Templates και static μέλη

- Απλή κλάση (όχι template class)
 - Τα **static** μέλη μοιράζονται από όλα τα αντικείμενα
- Class-template
 - Κάθε τύπος έχει δικά του αντίγραφα των **static** μεταβλητών
 - **static** μεταβλητές αρχικοποιούνται σε εμβέλεια αρχείου
 - Κάθε τύπος έχει δικά του αντίγραφα των **static** μεθόδων



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαια 18** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 7: C++ TEMPLATES, ΥΠΕΡΦΟΡΤΩΣΗ ΤΕΛΕΣΤΩΝ, ΕΞΑΙΡΕΣΕΙΣ

Υπερφόρτωση Τελεστών

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Υπερφόρτωση Τελεστών

Υπερφόρτωση Τελεστών

- Τύποι Τελεστών
 - Built in (**int**, **char**) ή ορισμένοι από τον χρήστη (user-defined)
 - Χρήση υπαρχόντων τελεστών με τύπους ορισμένους από τον χρήστη
- Υπερφόρτωση Τελεστών
 - Δημιουργία συνάρτησης για την κλάση
 - Το όνομα της συνάρτησης είναι **'operator'** ακολουθούμενο από το σύμβολο του τελεστή Π.χ:
 - **Operator+** για τον τελεστή πρόσθεσης +



Υπερφόρτωση Τελεστών σε Κλάση

- Χρήση τελεστών σε αντικείμενα κλάσεων
 - Πρέπει πρώτα να γίνει υπερφόρτωση τους για την κλάση
 - Εξαιρέσεις (δεν χρειάζεται υπερφόρτωση):
 - Τελεστής ανάθεσης, `=`
Memberwise ανάθεση μεταξύ αντικειμένων
 - Τελεστής διεύθυνσης, `&`
Επιστρέφει τη διεύθυνση του αντικειμένου
 - Μπορούν να υπερφορτωθούν αν θέλουμε διαφορετική συμπεριφορά (όπως και ο τελεστής κόμμα: `,`)
 - Προσφέρουν πιο σύντομη και κατανοητή διατύπωση:

```
object2 = object1.add(object2);
```

```
object2 = object2 + object1;
```



Διαθέσιμοι Τελεστές

Τελεστές που μπορούν να υπερφορτωθούν							
+	-	*	/	%	^	&	
~	!	=	<	>	+=	--=	*=
/=	%=	^=	&=	=	<<	>>	>>=
<<=	==	!=	<=	>=	&&		++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Τελεστές που δεν μπορούν να υπερφορτωθούν				
.	.*	::	?:	sizeof



Υπερφόρτωση Τελεστή ως μέλος κλάσης ή ως Friend συνάρτηση

■ Operator functions

○ Ως μέθοδος (μέλος της κλάσης)

- Το αριστερό μέλος του τελεστή είναι το αντικείμενο για το οποίο καλείται (μπορούμε να χρησιμοποιήσουμε και τον τελεστή `this` για να αναφερθούμε σε αυτό). Προφανώς είναι ιδίου τύπου με την κλάση στην οποία ορίζουμε τον τελεστή.
- Το δεξιό μέλος του τελεστή είναι όρισμα της συνάρτησης (οτιδήποτε τύπου θέλουμε)

○ Ως συνάρτηση (εκτός της κλάσης, όχι μέθοδος)

- Πρέπει να δοθούν και οι δύο παράμετροι (πριν και μετά τον τελεστή) ως ορίσματα
- Πρέπει να είναι friend συνάρτηση (προσθέτουμε σχετική δήλωση στην κλάση) για να προσπελάσει `private` ή `protected` μέλη

○ (), [], -> = πρέπει να οριστούν ως μέλη της κλάσης

○ Καλούνται

- Όταν το αριστερό μέλος του τελεστή (για binary τελεστές) είναι αντικείμενο της κλάσης
- Το μοναδικό όρισμα (για unary τελεστές) είναι αντικείμενο της κλάσης



Παράδειγμα

```
class myDate{
    friend ostream &operator<<( ostream &, const myDate & );
    friend istream &operator>>( istream &, myDate & );

public: myDate(int d=5, int m=5, int y=1994) :day(d), month(m), year(y) {}
    bool operator==( const myDate &right ) const;
    myDate& operator++(); myDate operator++( int );
    int operator[]( char i );

private:
    int day, month, year;
};
```

Υπερφόρτωση συναρτήσεων μη-μελών της κλάσης (απλά δήλωση ως friend)

Υπερφόρτωση συναρτήσεων μελών της κλάσης



Παράδειγμα

```
class myDate{
    friend ostream &operator<<( ostream &, const myDate & );
    friend istream &operator>>( istream &, myDate & );
public: myDate(int d=5, int m=5, int y=1994) :day(d), month(m), year(y) {}
    bool operator==( const myDate &right ) const;
    myDate& operator++(); myDate operator++( int );
    int operator[]( char i );
private:
    int day, month, year;
};

bool myDate::operator==( const myDate &right ) const{
    if ((day == right.day) && (month == right.month) && (year == right.year))
        return true;
    return false;
}

int myDate::operator[]( char i ){
    switch (i){
        case 'm':
            return month;
        case 'y':
            return year;
        case 'd':
            return day;
        default:
            return -1;
    }
}
```



Παράδειγμα

```
class myDate{
    friend ostream &operator<<( ostream &, const myDate & );
    friend istream &operator>>( istream &, myDate & );
public: myDate(int d=5, int m=5, int y=1994) :day(d), month(m), year(y) {}
    bool operator==( const myDate &right ) const;
    myDate& operator++(); myDate operator++( int );
    int operator[]( char i );
private:
    int day, month, year;
};
```

```
istream &operator>>( istream &input, myDate &a ){
    input >> a.year >> a.month >> a.day;
    return input;
}
```

```
ostream &operator<<( ostream &output, const myDate &a ){
    output << a.year << "/" << a.month << "/" << a.day;
    return output;
}
```

```
myDate &myDate::operator++() {
    day++; return *this;
}
```

```
myDate myDate::operator++( int ) {
    myDate temp = *this;
    day++;
    return temp;
}
```



Παράδειγμα

```
class myDate{
    friend ostream &operator<<( ostream &, const myDate & );
    friend istream &operator>>( istream &, myDate & );
public: myDate(int d=5, int m=5, int y=1994) :day(d), month(m), year(y) {}
    bool operator==( const myDate &right ) const;
    myDate& operator++(); myDate operator++( int );
    int operator[]( char i );
private:
    int day, month, year;
};

int main(){
    myDate dt1(15,7,2015);
    myDate dt2(25,9,2014);
    myDate dt3;
    cin >> dt3;
    cout << dt2 <<endl;
    cout << ++dt1; cout << dt1++ <<endl;
    cout << "Month: " << dt2['m'] <<endl;
    cout << ((dt1 == dt2)?"same date":"different date");
}
```



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 10** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 7: C++ TEMPLATES, ΥΠΕΡΦΟΡΤΩΣΗ ΤΕΛΕΣΤΩΝ, ΕΞΑΙΡΕΣΕΙΣ

Χειρισμός Εξαιρέσεων

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Χειρισμός Εξαιρέσεων

Εξαιρέσεις

- Εξαίρεση
 - Δείχνουν ότι κάποιο πρόβλημα προέκυψε στην εκτέλεση του προγράμματος
 - Κάτι μη φυσιολογικό
- Χειρισμός Εξαιρέσεων
 - Resolve exceptions
 - Το πρόγραμμα μπορεί να συνεχίσει την λειτουργία
 - Controlled termination
 - Για δημιουργία fault-tolerant programs



Try – Catch Blocks

- Κώδικας C++

```
try {  
    κώδικα που μπορεί να προκαλέσει εξαίρεση  
}  
catch (exceptionType) {  
    κώδικας για χειρισμό εξαιρέσεων  
}
```

- Το **try** μπλοκ περιέχει κώδικα ο οποίος μπορεί να προκαλέσει εξαίρεση
- Τα **catch** μπλοκ (1 ή περισσότερα):
 - Λαμβάνουν και χειρίζονται τις εξαιρέσεις
 - Μέσω παραμέτρου μπορούν να προσπελάσουν το αντικείμενο εξαίρεσης.



Παράδειγμα

```
#include <iostream>
#include <stdexcept>
using namespace std;
class DivideByZeroException : public out_of_range {
public:
    DivideByZeroException(int n, string m)
        : numerator(n), out_of_range(m) {}
    int numerator;
};

double divide( int numerator, int denominator ) {
    if ( denominator == 0 )
        throw DivideByZeroException(numerator,"attempted to divide with zero");
    return static_cast< double >( numerator ) / denominator;
}

int main(){
    int a,b;
    try{
        while ( cin >> a >> b )
            divide(a,b);
    }
    catch (out_of_range e){
        cout << e.what();
    }
}
```

Αν και δεν είναι υποχρεωτικό, εδώ φτιάχνουμε δικό μας τύπο εξαίρεσης, κληρονομώντας από τον υπάρχων τύπο `out_of_range`. Μέσω του δημιουργού μπορούμε να ορίσουμε την πληροφορία που θα αποθηκεύουμε στην εξαίρεση μέσω των ορισμάτων του. (πχ εδώ μπορούμε να κρατάμε τον αριθμητή της πράξης). Στον δημιουργό της κλάσης `out_of_range` μπορούμε να περνάμε μήνυμα σχετικό με το σφάλμα το οποίο θα είναι προσπελάσιμο μέσω της μεθόδου `what()` που έχει η κλάση `exception` και παράγωγες της.

Με κλήση του δημιουργού, δημιουργούμε εξαίρεση του τύπου που φτιάξαμε και την πετάμε με την εντολή `throw`.

Στο `catch` κομμάτι χειριζόμαστε εξαίρεσεις τύπου `out_of_range` ή παράγωγες της (όπως αυτή που φτιάξαμε).



Ροή εκτέλεσης

- Σημείο έγερσης εξαίρεσης (Throw point)
 - Το σημείο στο μπλοκ **try** όπου εγείρεται εξαίρεση
 - Αν χειριστεί η εξαίρεση
 - Το πρόγραμμα προσπερνάει τον υπόλοιπο κώδικα του **try** μπλοκ
 - Συνεχίζεται η λειτουργία μετά τα **catch** μπλοκ
 - αν δεν χειριστεί η εξαίρεση
 - Τερματίζεται η λειτουργία
- Αν δεν προκύψει εξαίρεση
 - Το πρόγραμμα αγνοεί τα **catch** blocks.



Τεχνικές Χειρισμού Εξαιρέσεων

- Αγνόηση εξαιρέσεων
 - Τυπικό για προσωπικό (όχι εμπορικό) λογισμικό
 - Το πρόγραμμα μπορεί να αποτυγχάνει
- Τερματισμός Προγράμματος
 - Συνήθως είναι κατάλληλο
 - Δεν είναι κατάλληλο για κρίσιμες εφαρμογές
- Set error indicators
 - Unfortunately, may not test for these when necessary
- Test for error condition
 - Call `exit (<cstdliblib>)` and pass error code



Άμεση έγερση εξαιρέσεων

- Εντολή **throw**
 - Εγείρει μια εξαίρεση
 - Χρήση όταν προκύπτει το σφάλμα
 - Μπορούμε με «πετάξουμε» με την `throw` σχεδόν οτιδήποτε (αντικείμενα-εξαιρέσεις, built-in τύπους όπως `integer`, κτλ.)
 - `throw myObject;`
 - `throw 5;`
- Αντικείμενα-Εξαιρέσεις
 - Έχουν ως κλάση βάσης την `exception` (`<exception>`)
 - Ο δημιουργός μπορεί να έχει ένα αλφαριθμητικό για περιγραφή του σφάλματος.
 - Η περιγραφή μπορεί να ανακτηθεί μέσω της μεθόδου `what ()`



Επανεγερση

- Επανεγερση εξαίρεσης (rethrowing)
 - Χρησιμοποιείται σε `catch` μπλοκ, όταν δεν μπορεί να χειριστεί η εξαίρεση ώστε να επανεγερθεί
 - Can rethrow exception to another handler
 - Goes to next enclosing `try` block
 - Corresponding `catch` blocks try to handle
- Για να ξαναπετάξουμε εξαίρεση μέσα σε `catch block` καλούμε την εντολή `throw` (χωρίς όρισμα)



Λίστα εξαιρέσεων συνάρτησης

- Ορίζουμε την λίστα των εξαιρέσεων που μπορεί να εγερθούν στην συνάρτηση αυτήν
 - Also called throw list

```
int someFunction( double value )
    throw ( ExceptionA, ExceptionB, ExceptionC )
{
    // σώμα συνάρτησης
}
```
 - Η συνάρτηση μπορεί να πετάξει εξαιρέσεις τύπου **ExceptionA**, **ExceptionB**, and **ExceptionC** (ή παράγωγες αυτών)
 - Αν προκύψει άλλου είδους εξαίρεση (και δεν χειριστεί μέσα στο σώμα της συνάρτησης με κάποιο catch block) προκύπτει απροσδόκητο σφάλμα και τερματίζει το πρόγραμμα
 - Αν δεν ορίσουμε λίστα throw, μπορεί να πετάξει οποιαδήποτε
 - Αν ορίσουμε κενή λίστα throw, δεν μπορεί να πετάξει καμία εξαίρεση



Παραδείγματα

```
void function() throw(int){
    throw 5;
}

int main(){
    try{ function(); }
    catch(int){
        cout << " handled";
    }
}
```

handled

(εγείρεται εξαίρεση αποδεκτού τύπου στην function).
Η main έχει κατάλληλο catch μπλοκ χειρισμού του τύπου οπότε χειρίζεται και τερματίζει κανονικά η λειτουργία

```
void function() throw(int){
    throw 'k';
}

int main(){
    try{ function(); }
    catch(int){
        cout << " handled";
    }
}
```

Τερματισμός με σφάλμα

Μη αποδεκτός τύπος εξαίρεσης.
Δεν έχει οριστεί ο τύπος char στην λίστα throw της συνάρτησης.

```
void function() throw(int){
    try{
        throw 'k';
    }
    catch(...){
        cout << "handled internally";
    }
}

int main(){
    try{ function(); }
    catch(int){
        cout << " handled";
    }
}
```

handled internally

(Εγείρεται εξαίρεση μη αποδεκτού τύπου, ωστόσο χειρίζεται εσωτερικά).
Η main δεν «ενημερώνεται» ότι προέκυψε εξαίρεση στην function



Παραδείγματα

```
void function() throw(int, char){
    try{
        throw 'k';
    }
    catch(...){
        cout << "handled internally";
        throw;
    }
}

int main(){
    try{ function(); }
    catch(int){
        cout << " handled";
    }
}
```

handled internally

Τερματίζει με σφάλμα

(Η εξαίρεση συλλαμβάνεται αρχικά εσωτερικά, ωστόσο με την throw την πετάει ξανά στο παραπάνω επίπεδο (main) , το οποίο δεν έχει κατάλληλο catch χειρισμού της.

```
void function() throw(int, char){
    try{
        throw 'k';
    }
    catch(...){
        cout << "handled internally";
        throw;
    }
}

int main(){
    try{ function(); }
    catch(char ch){
        cout <<endl<< ch << " handled";
    }
}
```

handled internally

k handled

Η εξαίρεση χειρίζεται και εσωτερικά και στην main



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 17** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 8: C++ ΒΙΒΛΙΟΘΗΚΗ STL, ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Βιβλιοθήκη STL

Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Βιβλιοθήκη STL

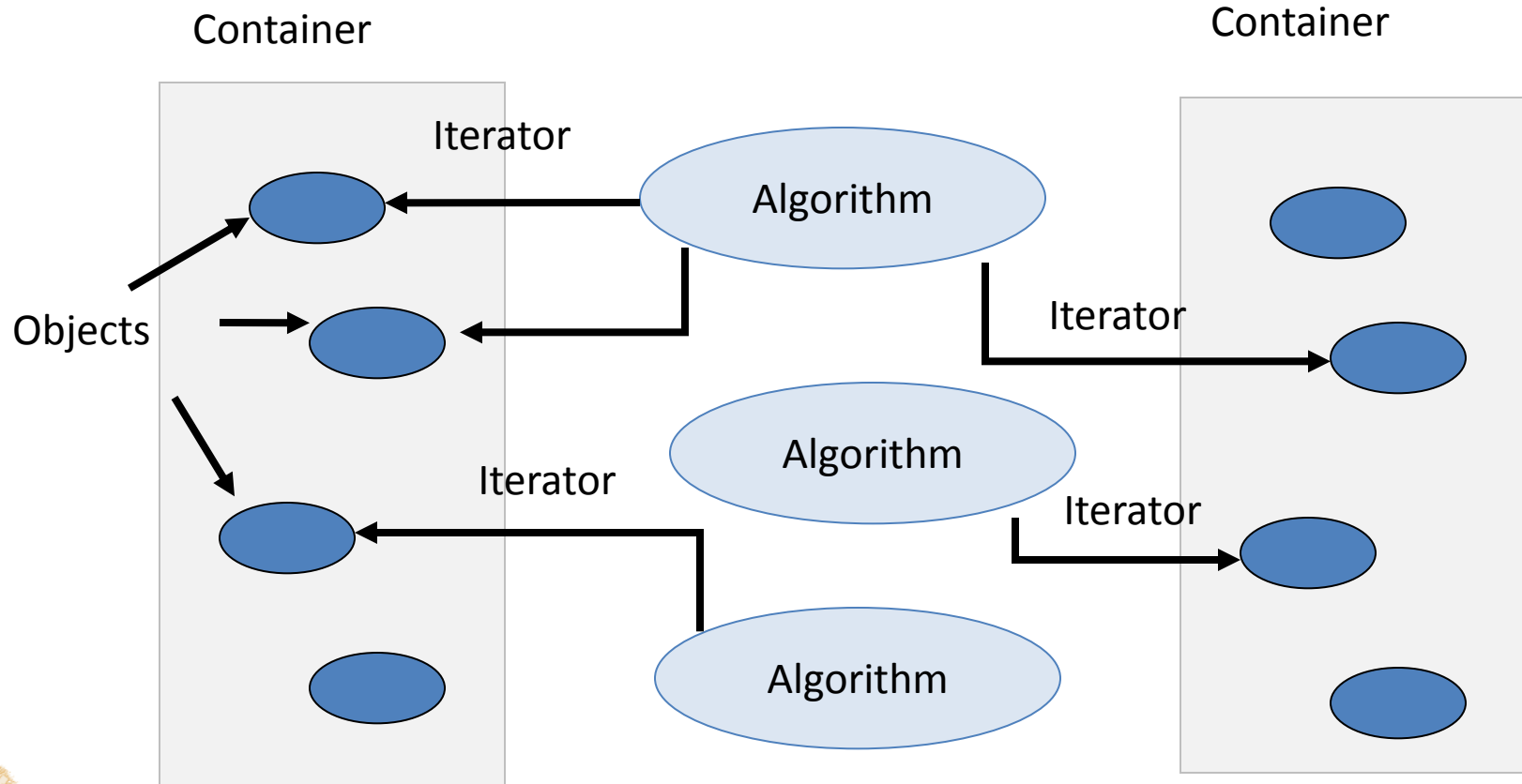
Standard Template Library

- Η standard template library (STL) αποτελείται
 - Containers
 - Algorithms
 - Iterators
- Ένας **container** ένας τρόπος οργάνωσης μίας συλλογής αντικειμένων στην μνήμη
- **Algorithms** στην STL είναι διαδικασίες που εφαρμόζονται στους containers για να επεξεργαστούν τα δεδομένα τους, πχ. Αναζήτηση αντικειμένου
- **Iterators** είναι μία γενίκευση των δεικτών (pointers), δείχνουν αντικείμενα μέσα σε έναν container



Containers, Iterators, Algorithms

Οι αλγόριθμοι χρησιμοποιούν iterators για να αλληλεπιδράσουν
Με Αντικείμενα των containers



Containers

- Ένας **container** είναι ένας τρόπος να αποθηκεύσουμε δεδομένα, είτε βασικούς τύπους είτε αντικείμενα κλάσεων.
- Η STL παρέχει διάφορους βασικούς τύπους containers
 - `<vector>` : one-dimensional array
 - `<list>` : double linked list
 - `<deque>` : double-ended queue
 - `<queue>` : queue
 - `<stack>` : stack
 - `<set>` : set
 - `<map>` : associative array



Sequence Containers

- Ένας **sequence container** αποθηκεύει ένα σύνολο αντικειμένων στην ακολουθία , με άλλα λόγια κάθε αντικείμενο (εκτός από το πρώτο και το τελευταίο) ακολουθείται από ένα συγκεκριμένο αντικείμενο: `<vector>`, `<list>` and `<deque>`
- Οι κλασικοί C++ πίνακες με σταθερό μέγεθος δεν μεταβάλλονται κατά την εκτέλεση έχουν το πλεονέκτημα της τυχαίας προσπέλασης
- `<vector>` είναι επεκτάσιμος τύπος αλλά οι εισαγωγές/διαγραφές στο μέσω είναι αργά.



Sequence Containers

- `<list>` είναι διπλά διασυνδεδεμένη λίστα και είναι γρήγορη η εισαγωγή/διαγραφή αλλά αργή η προσπέλαση
- `<deque>` είναι ουρά δύο άκρων, δηλαδή εισάγει και διαγράφει στοιχεία από τα δύο άκρα → `stack + queue + list`

Associative Containers

- Ένα **associative container** είναι **non-sequential** και χρησιμοποιεί ένα κλειδί (*key*) για την προσπέλαση των αντικειμένων. Τα κλειδιά είναι αριθμοί ή συμβολοσειρές χρησιμοποιούνται από τον **container** για να διαχειριστεί τα **αντικείμενα** με μία σειρά , πχ λεξικό.



Associative Containers

- Ένα **<set>** αποθηκεύει ένα αριθμό αντικειμένων που περιέχουν κλειδιά. Τα κλειδιά είναι τα χαρακτηριστικά που χρησιμοποιούνται για την διάταξη των στοιχείων.

Πχ. Ένα `set` μπορεί να αποθηκεύει αντικείμενα της κλάσης

`Person` που διατάσσονται αλφαβητικά ανάλογα με το όνομα τους

- Ένα **<map>** αποθηκεύει ζευγάρια αντικειμένων : ένα **κλειδί** και μία συσχετιζόμενη **τιμή**. Ένα `<map>` είναι αντίστοιχο με έναν πίνακα αλλά αντί για αριθμούς δείκτες μπορούν να χρησιμοποιηθούν οποιοδήποτε τύπου αντικείμενα
- `<set>` και `<map>` επιτρέπουν ένα κλειδί για κάθε τιμή ενώ τα `<multiset>` και `<multimap>` επιτρέπουν πολλές εμφανίσεις



Vector Container

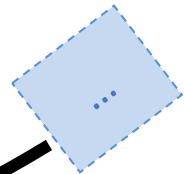
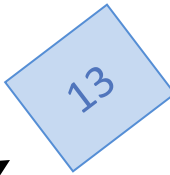
```
int array[5] = {12, 7, 9, 21, 13};  
vector<int> v(array,array+5);
```



`v.pop_back();`



`v.push_back(15);`



0 1 2 3 4



`v.begin();`

An upward-pointing arrow from the text `v.begin();` to the first box (index 0) of the vector.

`v[3]`

An upward-pointing arrow from the text `v[3]` to the fourth box (index 3) of the vector.



Vector Container

```
#include <vector>
#include <iostream>
```

```
vector<int> v(3);
```

```
v[0]=23;
```

```
v[1]=12;
```

```
v[2]=9; // πλήρης
```

```
v.push_back(17); // προσθήκη νέου στοιχείου στο τέλος
```

```
for (int i=0; i<v.size(); i++)
```

```
    cout << v[i] << " "; // random access - στοιχείο i
```

```
cout << endl;
```



Vector Container

```
#include <vector>
#include <iostream>

int arr[] = { 12, 3, 17, 8 }; // απλός πίνακας
vector<int> v(arr, arr+4);    // αρχικοποίηση vector από πίνακα

while ( ! v.empty() ) {     // Μέχρι να αδειάσει
    cout << v.back() << " "; // εκτύπωση του τελευταίου στοιχείου
    v.pop_back();           // αφαίρεση του τελευταίου στοιχείου
}
cout << endl;
```



Δημιουργοί Vector

- Ένας vector μπορεί να αρχικοποιηθεί δίνοντας το μέγεθος του και τον τύπο του αντικειμένου (prototype) ή άλλον vector.

```
vector<Date> x(1000); // δημιουργεί vector μεγέθους 1000  
                    // απαιτεί ύπαρξη default constructor στην Date
```

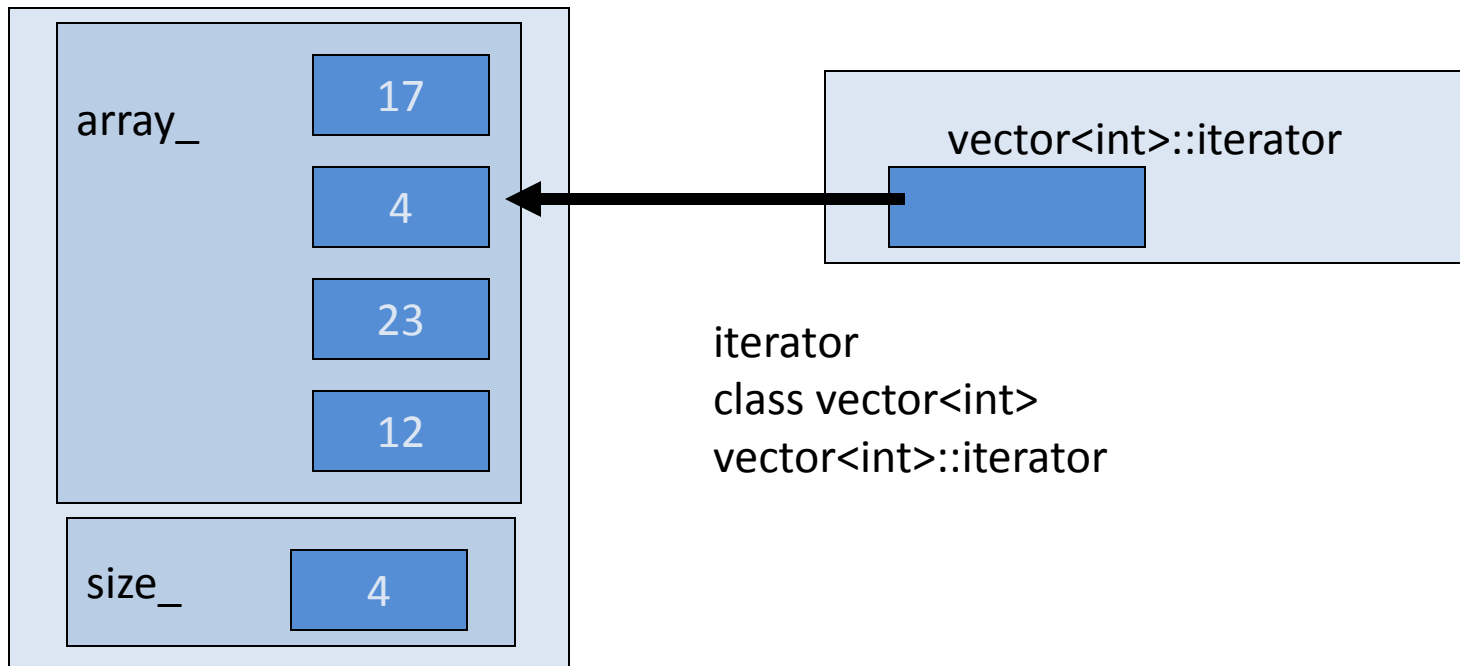
```
vector<Date> dates(10, Date(17, 12, 1999));  
                // αρχικοποίηση όλων σε 17.12.1999
```

```
vector<Date> y(x); // αρχικοποίηση του vector y με τον vector x
```



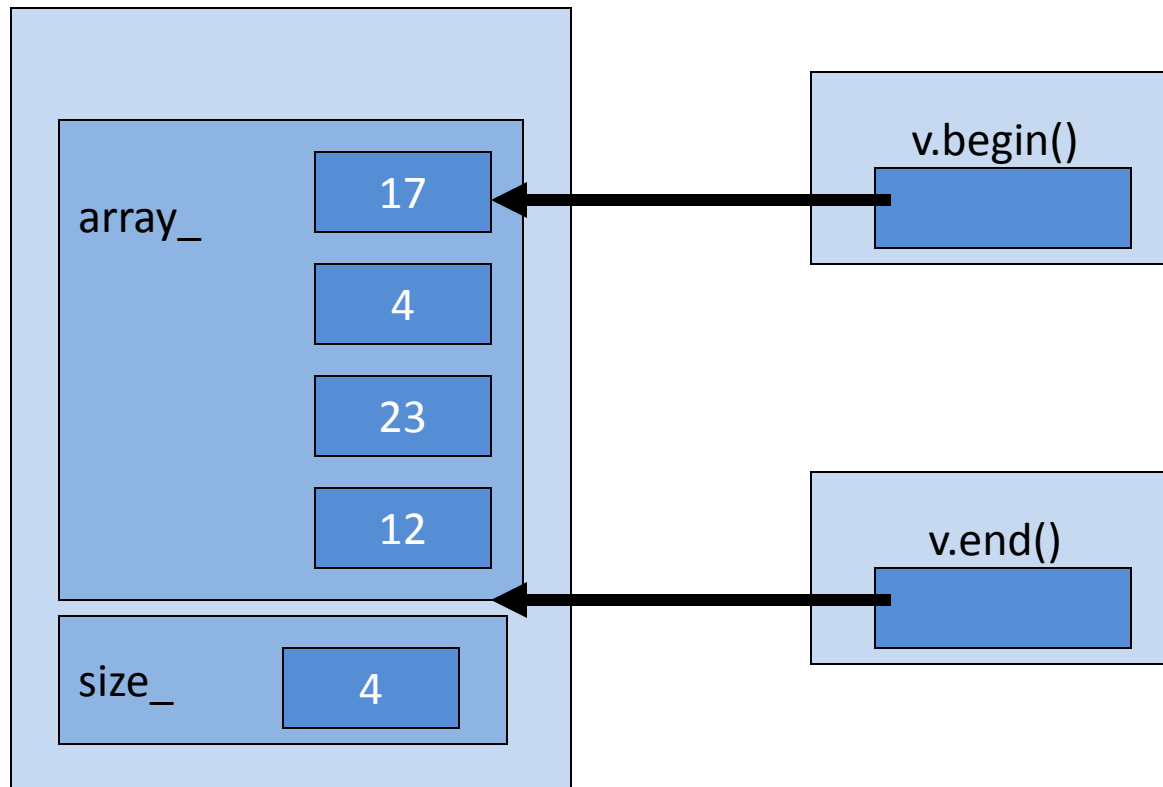
Iterators

- **Iterators** είναι **pointer-like** αντικείμενα που χρησιμοποιούνται για την **προσπέλαση** αντικειμένων σε έναν **container**.
- Χρησιμοποιούνται για να κινούνται σειριακά από αντικείμενο σε αντικείμενο → *iterating* διαμέσου ενός container.



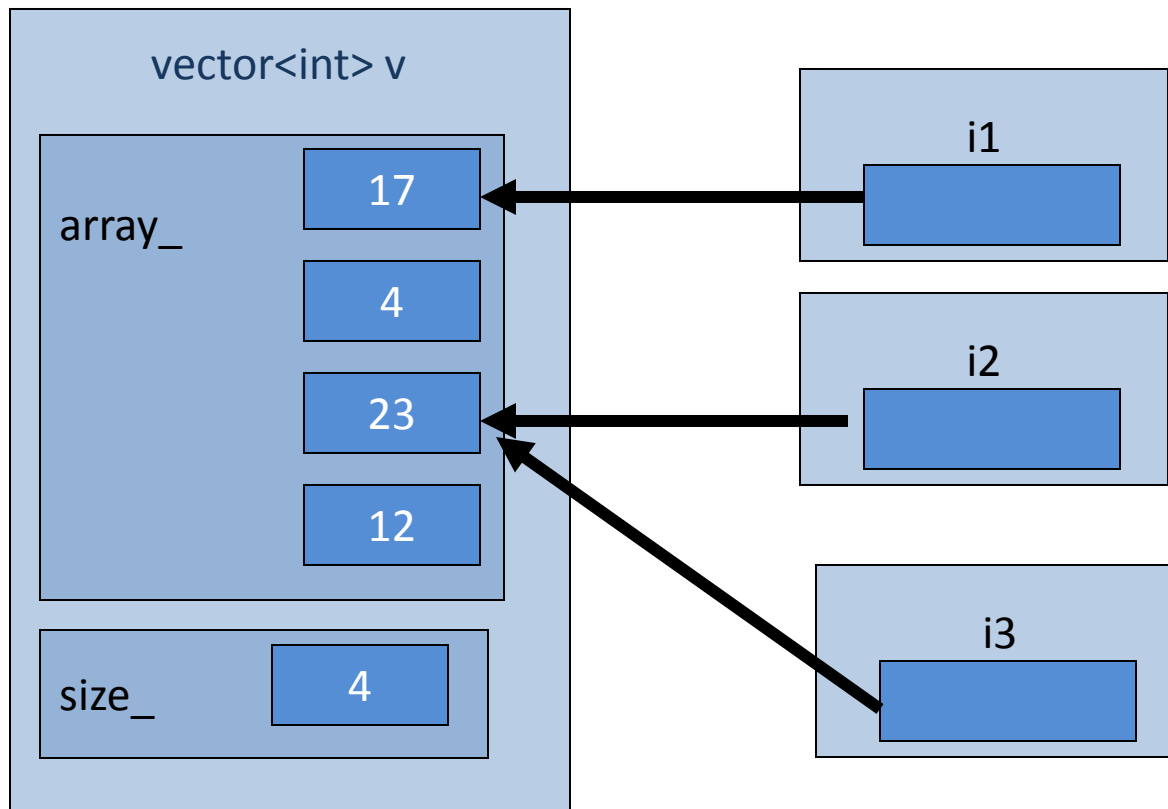
Iterators

- Οι member functions `begin()` και `end()` επιστρέφουν έναν iterator στο πρώτο και τελευταίο στοιχείο του container



Iterators

- Μπορούμε να έχουμε πολλαπλούς iterators ταυτόχρονα



Iterators

```
#include <vector>
#include <iostream>

int arr[] = { 12, 3, 17, 8 };
vector<int> v(arr, arr+4);

vector<int>::iterator iter=v.begin(); // δήλωση iterator και ανάθεση ώστε να
//δείχνει στο πρώτο στοιχείο του vector

cout << "first element of v=" << *iter; // Με τον τελεστή * παίρνουμε το
// στοιχείο που δείχνει ο iterator

iter++; // ο iterator δείχνει τώρα στο επόμενο στοιχείο

iter=v.end()-1; // ο iterator δείχνει τώρα στο τελευταίο στοιχείο
```



Iterators

```
int max(vector<int>::iterator start, vector<int>::iterator end) {
    int m=*start;
    while(start != stop) {
        if (*start > m) m=*start; ++start;
    }
    return m;
}
```

```
cout << "max of v = " << max(v.begin(),v.end());
```



Iterators

```
#include <vector>
#include <iostream>

int arr[] = { 12, 3, 17, 8 };
vector<int> v(arr, arr+4);

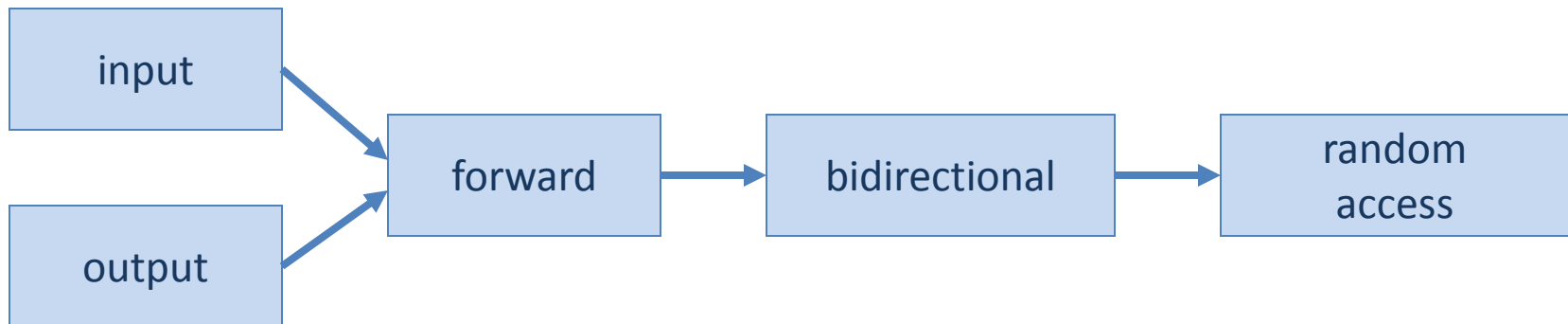
for (vector<int>::iterator i=v.begin(); i!=v.end(); i++) {
    cout << *i << " ";
}

cout << endl;
```



Κατηγορίες Iterator

- Δεν είναι δυνατό κάθε iterator να χρησιμοποιηθεί με κάθε container πχ. η κλάση λίστα δεν δίνει random access iterator
- Κάθε αλγόριθμος απαιτεί έναν iterator με κάποιες συγκεκριμένες ιδιότητες πχ. Να χρησιμοποιεί το [] τελεστή για random access
- Οι Iterators χωρίζονται σε 5 κατηγορίες όπου κάθε κατηγορία εμπεριέχει την κατώτερη της.



For_Each() Algorithm

```
#include <vector> #include <algorithm>
#include <iostream>
```

```
void show(int n) {
    cout << n << " ";
}
```

```
int arr[] = { 12, 3, 17, 8 };
vector<int> v(arr, arr+4);
```

```
for_each (v.begin(), v.end(), show);
// εφαρμογή της συνάρτησης show
// σε κάθε στοιχείο του vector v
```



Find() Algorithm

```
#include <vector> #include <algorithm>
#include <iostream>

int key;
int arr[] = { 12, 3, 17, 8, 34, 56, 9 };
vector<int> v(arr, arr+7);
vector<int>::iterator iter;

cout << "enter value :";
cin >> key;
iter=find(v.begin(),v.end(),key); // finds integer key in v

if (iter != v.end())
    cout << "Element " << key << " found" << endl;
else
    cout << "Element " << key << " not in vector v" << endl;
```



Find_If() Algorithm

```
#include <vector>
#include <algorithm>
#include <iostream>

Bool mytest(int n) {
    return (n>21) && (n <36);
};

int arr[] = { 12, 3, 17, 8, 34, 56, 9 };
vector<int> v(arr, arr+7);
vector<int>::iterator iter;

iter=find_if(v.begin(),v.end(),mytest);
    // βρίσκει στοιχείο στον v για το οποίο η mytest επιστρέφει true

if (iter != v.end())
    cout << "found " << *iter << endl;
else
    cout << "not found" << endl;
```



Count_If() Algorithm

```
#include <vector>
#include <algorithm>
#include <iostream>

Bool mytest(int n) {
    return (n>14) && (n <36);
};

int arr[] = { 12, 3, 17, 8, 34, 56, 9 };
vector<int> v(arr, arr+7);

int n=count_if(v.begin(),v.end(),mytest);
    // μετράει για πόσα στοιχεία του v η mytest επιστρέφει true

cout << "found " << n << " elements" << endl;
```



List Container

- Ένας STL list container είναι μία διπλά διασυνδεδεμένη λίστα: κάθε αντικείμενο δείχνει στον προηγούμενο και στον επόμενο
- Είναι δυνατό να προσθέσουμε/αφαιρέσουμε στοιχεία από τα δύο άκρα της λίστας
- Δεν επιτρέπει random access αλλά μπορούμε να εισάγουμε/διαγράψουμε από την μέση καθώς και να συνενώσουμε και να διατάξουμε



List Container

```
int array[5] = {12, 7, 9, 21, 13};  
list<int> li(array,array+5);
```



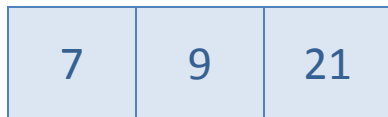
li.pop_back();



li.push_back(15);



li.pop_front();



li.push_front(8);



li.insert()



Insert Iterators

- Η χρήση του copy algorithm διαγράφει τα υπάρχοντα στοιχεία

```
#include <list>
int arr1[] = { 1, 3, 5, 7, 9 };
int arr2[] = { 2, 4, 6, 8, 10 };
list<int> l1(arr1, arr1+5);
list<int> l2(arr2, arr2+5);
copy(l1.begin(), l1.end(), l2.begin());
// αντιγραφή των στοιχείων του l1 στο l2 (τα στοιχεία του l2 διαγράφονται)
// l2 = { 1, 3, 5, 7, 9 }
```



Insert Iterators

- Οι operators διαφοροποιούν τη συμπεριφορά του copy algorithm
 - back_inserter : εισάγει νέο στοιχείο στο τέλος
 - front_inserter : εισάγει νέο στοιχείο στην αρχή
 - inserter : εισάγει νέο στοιχείο σε θέση

```
#include <list>
int arr1[]={ 1, 3, 5, 7, 9 };
int arr2[]={ 2, 4, 6, 8, 10 };
list<int> l1(arr1, arr1+5);
list<int> l2(arr2, arr2+5);
copy(l1.begin(), l1.end(), back_inserter(l2)); // χρήση back_inserter
// adds contents of l1 to the end of l2 = { 2, 4, 6, 8, 10, 1, 3, 5, 7, 9 }
copy(l1.begin(), l1.end(), front_inserter(l2)); // use front_inserter
// adds contents of l1 to the front of l2 = { 9, 7, 5, 3, 1, 2, 4, 6, 8, 10 }
copy(l1.begin(), l1.end(), inserter(l2,l2.begin()));
// adds contents of l1 at the "old" beginning of l2 = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 10 }
```



Sort & Merge

- Sort και merge

```
#include <list>

int arr1[] = { 6, 4, 9, 1, 7 };
int arr2[] = { 4, 2, 1, 3, 8 };
list<int> l1(arr1, arr1+5);
list<int> l2(arr2, arr2+5);
l1.sort(); // l1 = {1, 4, 6, 7, 9}
l2.sort(); // l2 = {1, 2, 3, 4, 8}
l1.merge(l2); // merges l2 into l1
// l1 = { 1, 1, 2, 3, 4, 4, 6, 7, 8, 9}, l2 = {}
```



Functions Objects

- `sort`, `merge`, `accumulate` μπορούν να πάρουν σαν ορίσματα ένα `function object`
- Το `function object` είναι ένα αντικείμενο μιας `template class` που έχει ένα `single member function` : the overloaded operator `()`
- Είναι δυνατό να χρησιμοποιηθούν `user-written functions` στη θέση των `pre-defined function objects`

```
#include <list> #include <functional>
int arr1[] = { 6, 4, 9, 1, 7 };
list<int> l1(arr1, arr1+5);
```

```
l1.sort(greater<int>()); // χρήση function object greater<int>
// για ταξινόμηση αντίστροφης σειράς l1 = { 9, 7, 6, 4, 1 }
```



Function Objects

- Ο algorithm `accumulate` συναθροίζει δεδομένα πάνω από τα στοιχεία πχ `sum of elements`

```
#include <list>
#include <functional>
#include <numeric>

int arr1[] = { 6, 4, 9, 1, 7 };
list<int> l1(arr1, arr1+5); // αρχικοποίηση l1 απο arr1

int sum = accumulate(l1.begin(), l1.end(), 0, plus<int>());
int sum = accumulate(l1.begin(), l1.end(), 0); // ισοδύναμο
int fac = accumulate(l1.begin(), l1.end(), 0, times<int>());
```



User Defined Function Objects

```
class squared_sum{  
public:  
    int operator() (int n1, int n2) {  
        return n1+n2*n2;  
    }  
};
```

```
int sq = accumulate(l1.begin(), l1.end() , 0, squared_sum() );  
        // computes the sum of squares
```



User Defined Function Objects

```
template <class T>
class squared_sum {
public:
    T operator() (T n1, T n2) {
        return n1+n2*n2;
    }
};

vector<complex> vc;
complex sum_vc;
vc.push_back(complex(2,3));
vc.push_back(complex(1,5));
vc.push_back(complex(-2,4));
sum_vc = accumulate(vc.begin(), vc.end(), complex(0,0), squared_sum<complex>() );
// computes the sum of squares of a vector of complex numbers
```



Associative Containers

- Σε ένα associative container τα αντικείμενα δεν σχηματίζουν ακολουθία αλλά δενδρικές δομές ή hash table.
- Τα προτερήματα τους είναι η ταχύτητα αναζήτησης (δυναμική κλπ)
- Η αναζήτηση γίνεται με βάση το κλειδί που είναι είτε αριθμός είτε συμβολοσειρά
- Η *value* είναι ένα χαρακτηριστικό των objects στο container
- Η STL έχει δύο βασικά associative containers
 - sets and multisets
 - maps and multimaps



Sets and Multisets

```
#include <set>
string names[] = {"Yiannis", "Takis", "Petros", "Christos", "Anestis"};

set<string, less<string> > nameSet (names, names+5);
    // δημιουργία σετ με ονόματα αλφαβητικά ταξινομημένα

nameSet.insert("Eleni"); // εισαγωγή νέων στοιχείων
nameSet.insert("Maria");
nameSet.erase("Petros"); // αφαίρεση στοιχείου
set<string, less<string> >::iterator iter; // set iterator
string searchname;
cin >> searchname;
iter=nameSet.find(searchname); // εύρεση του ονόματος στο σετ

if (iter == nameSet.end()) // έλεγχος αν ο iterator δείχνει στο τέλος
    cout << searchname << " not in set!" <<endl;
else
    cout << searchname << " is in set!" <<endl;
```



Set and Multisets

```
string names[]={ "Yiannis", "Takis", "Petros", "Christos", "Anestis" };

set<string, less<string> > nameSet (names, names+5);
set<string, less<string> >::iterator iter; // set iterator

iter=nameSet.lower_bound("K");
    // set iterator to lower start value "K"

while (iter != nameSet.upper_bound("Z"))
    cout << *iter++ << endl;
// displays Petros, Takis, Yiannis
```



Maps and Multimaps

- Ένα map αποθηκεύει pairs `<key, value>` ενός key object και ενός associated value object.
- Το key object περιέχει ένα key με το οποίο αναζητούμε και το value object περιέχει τις πληροφορίες
- Το key μπορεί να είναι string, πχ όνομα, ΑΣΤ, ή αριθμός



Maps and Multimaps

```
#include <map>
string names[]= {"Yiannis", "Takis", "Petros", "Christos", "Anestis"};
int numbers[]= {62622, 56879, 12587, 387546, 98794};

map<string, int, less<string> > phonebook; map<string, int,
less<string> >::iterator iter;

for (int j=0; j<5; j++)
    phonebook[names[j]]=numbers[j]; // αρχικοποίηση

for (iter = phonebook.begin(); iter !=phonebook.end(); iter++)
    cout << (*iter).first << " : " << (*iter).second << endl;
cout << "Phone of Yiannis is: " << phonebook["Yiannis"] << endl;
```



Professor Class

```
Class Professor {  
    private: string lastName;  
             string firstName;  
             long  phoneNumber;  
  
    public:  
        person(string lana, string fina, long pho) : lastName(lana),  
                                                    firstName(fina), phonenumber(pho) {}  
  
        bool operator<(const person& p);  
        bool operator==(const person& p);  
}
```



Maps & Multimaps

```
Professor p1 ("Ioannis", "Hatzis", 284545632);  
Professor p2 ("Christos", "Makris", 226874521);  
Professor p3 ("Nikos", "Ioannou", 284586541);
```

```
multiset< Professor, less< Professor >> profSet;  
multiset< Professor, less< Professor >>::iterator iter;
```

```
profSet.insert(p1);  
profSet.insert(p2);  
profSet.insert(p3);
```



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από τα **Κεφάλαια 15, 16** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
[http://media.pearsoncmg.com/ph/esm/deitel/cpp htp 9/code examples/Code Examples.zip](http://media.pearsoncmg.com/ph/esm/deitel/cpp_htp_9/code_examples/Code_Examples.zip)



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 8: C++ ΒΙΒΛΙΟΘΗΚΗ STL, ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Δομές Δεδομένων

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Δομές Δεδομένων

Εισαγωγή

- Δομές σταθερού μεγέθους
 - Arrays, structs
- Δυναμικές δομές δεδομένων
 - Το μέγεθος αυξάνεται και μειώνεται κατά την εκτέλεση του προγράμματος
 - Διασυνδεδεμένες λίστες (Linked lists)
 - Είσοδος και αφαίρεση από οποιοδήποτε σημείο
 - Στοίβες (Stacks)
 - Είσοδος και αφαίρεση από την κορυφή
 - Ουρές (Queues)
 - Είσοδος από το τέλος, αφαίρεση από την αρχή
 - Δυαδικά Δέντρα (Binary trees)
 - Αποδοτική αναζήτηση/ταξινόμηση



Αυτό-αναφερόμενη κλάση

- Αυτό-αναφερόμενες κλάσεις (Self-referential)
 - Έχουν δείκτη σε αντικείμενο της ίδια κλάσης
 - Σύνδεση μεταξύ τους για σχηματισμό χρήσιμων δομών
 - Lists, stacks, queues, trees
 - Η δομή τερματίζεται με το τελευταίο στοιχείο να δείχνει στο **NULL**

```
class Node {  
    public:  
        Node( int );  
    private:  
        int data;  
        Node *nextPtr;  
};
```



Δέσμευση/Αποδέσμευση Μνήμης

- Δυναμική δέσμευση μνήμης
 - Δέσμευση και απελευθέρωση μνήμης κατά την εκτέλεση
 - Δημιουργία και αφαίρεση κόμβων
- Τελεστής **new**
 - Επιστρέφει δείκτη στο νέο αντικείμενο που δημιουργεί
 - `Node *newPtr = new Node(5);`
 - Πετάει εξαίρεση `bad_alloc` αν δεν υπάρχει διαθέσιμη μνήμη
- Τελεστής **delete**
 - Αποδεσμεύει την μνήμη που δεσμεύτηκε κατά την δημιουργία
 - `Delete newPtr;`
 - Προσοχή: Δεν διαγράφεται ο δείκτης, απλά αποδεσμεύεται η θέση στην οποία εξακολουθεί να δείχνει.



Διασυνδεδεμένη Λίστα

- Διατηρεί δείκτη στο πρώτο στοιχείο (Node) της λίστας
 - Κάθε στοιχείο διαθέτει δείκτη προς το επόμενο οπότε μπορούμε να τα διαπεράσουμε σειριακά
- Το τελευταίο στοιχείο δείχνει στο null (zero), σηματοδοτώντας ότι φτάσαμε στο τέλος της λίστας
- Δυναμική Λίστα
 - Κόμβου προστίθενται και αφαιρούνται ανάλογα τις ανάγκες
 - Η τιμή που αποθηκεύεται στον κόμβο (value) μπορεί να είναι οποιοδήποτε τύπου
- Λειτουργίες:
 - Είσοδος κόμβου στην αρχή
 - Είσοδος κόμβου στο τέλος
 - Αφαίρεση κόμβου από την αρχή
 - Αφαίρεση κόμβου από το τέλος



Τύποι Διασυνδεδεμένων Λιστών

- Τύποι Διασυνδεδεμένων Λιστών
 - Μονά συνδεδεμένη Λίστα
 - Pointer to first node
 - Διαπέραση προς μια κατεύθυνση μόνο (το τελευταίο δείχνει στο null)
 - Κυκλική, μονά συνδεδεμένη
 - Όμοια, αλλά το τελευταίο δείχνει στο πρώτο
 - Διπλά συνδεδεμένη
 - Κάθε κόμβος έχει δείκτη στο επόμενο και στο προηγούμενο στοιχείο
 - Διαπέραση και προς τις δύο κατευθύνσεις
 - Ο τελευταίος κόμβος δείχνει στο null ως επόμενο
 - Ο πρώτος κόμβος δείχνει στο null ως προηγούμενο
 - Κυκλική, διπλά συνδεδεμένη
 - Όμοια, αλλά το πρώτο δείχνει στο τελευταίο και αντόστροφα



Στοίβα

- Στοίβα (Stack)
 - Κόμβοι προστίθενται και αφαιρούνται από την κορυφή
 - Περιορισμένη έκδοση διασυνδεδεμένης λίστας
 - Δομή: Last-in, first-out (LIFO)
 - Ο πάτος της στοίβας δείχνει στο null
- Λειτουργίες
 - Push: προσθήκη κόμβου στην κορυφή
 - Pop: Αφαίρεση κόμβου από την κορυφή



Υλοποίηση στοίβας

```
class Node{
private:
    int value;
    Node* next;
public:
    Node (int valueIn, Node* nextIn): value(valueIn), next(nextIn) {}
    ~Node() {} int getValue() const {return value;}
    Node* getNext() const {return next;}
};
```

```
class Stack{
private:
    Node* top;
public:
    Stack(){ top=0; }
    ~Stack() {
        Node* A;
        while(top!=0){
            A=top;
            top=top->getNext();
            delete A;
        }
    }
    void push(int k){
        Node* n = new Node(k,top);
        top = n;
    }
    int pop(){
        if (top==0)
            return 0;
        Node* A=top;
        int k=top->getValue();
        top=top->getNext();
        delete A;
        return k;
    }
};
```

```
int main() {
    Stack s;
    s.push(1);
    s.push(2);
    s.push(3);
    cout<< s.pop(); // 3
    cout<< s.pop(); // 2
    cout<< s.pop(); // 1
    cout<< s.pop(); // 0
    return 0;
}
```



Ουρά

- Ουρά (Queue)
 - Σαν γραμμή αναμονής
 - Είσοδος στο τέλος(ουρά) , αφαίρεση από την αρχή.
 - Δομή: First-in, first-out (FIFO)
- Λειτουργίες
 - Enqueue: Κόμβοι προστίθενται στο τέλος (*ουρά - tail*)
 - Dequeue: αφαίρεση από μπροστά (*κεφαλή - head*)



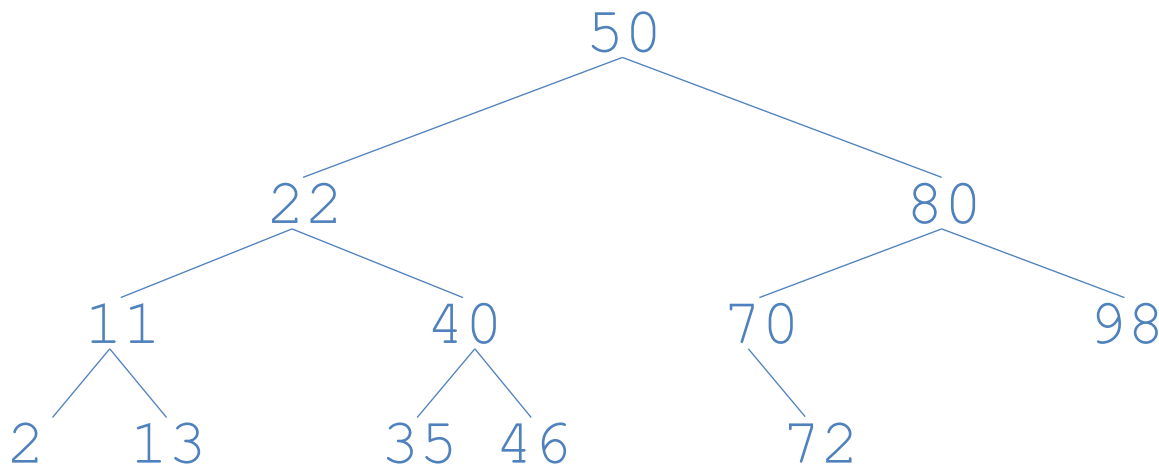
Δέντρα

- Δέντρα - Trees
 - Μη-γραμμικές δομές δύο διαστάσεων
 - Οι κόμβοι έχουν δύο ή περισσότερες συνδέσεις
 - Τα δυαδικά έχουν ακριβώς 2 συνδέσεις/δείκτες από τους οποίους μπορεί ένας ή και οι δύο να είναι null (π.χ φύλλα του δέντρου)
- Ορολογία
 - Ρίζα – Root: ο πρώτος κόμβος του δέντρου
 - Οι συνδέσεις/δείκτες ενός κόμβου αναφέρονται στα παιδιά του
 - Φύλλο – Leaf: κόμβος χωρίς παιδιά



Δυαδικό Δέντρο Αναζήτησης

- Δυαδικό δέντρο αναζήτησης
 - Οι τιμές στο αριστερό υπο-δέντρο κάθε κόμβου είναι μικρότερες από την τιμή του πατέρα
 - Οι τιμές στο δεξιό υπο-δέντρο κάθε κόμβου είναι μεγαλύτερες από την τιμή του πατέρα
 - Δεν επιτρέπονται διπλές τιμές
 - Γρήγορη αναζήτηση, $\log_2 n$ συγκρίσεις για ισορροπημένα δέντρα



Δυαδικό Δέντρο Αναζήτησης

- Είσοδος κόμβων

- Χρήση αναδρομικής συνάρτησης
- Ξεκινάει από την ρίζα
- Αν ο τρέχον κόμβος είναι άδειος, εισαγωγή του κόμβου σε αυτόν (τέλος αναδρομής)
- Αλλιώς,
 - Αν η τιμή του νέου κόμβου είναι μεγαλύτερη από του τρέχοντα κόμβου, εισαγωγή στο δεξιά υποδέντρο (αναδρομική κλήση)
 - Αν η τιμή του νέου κόμβου είναι μικρότερη από του τρέχοντα κόμβου, εισαγωγή στο αριστερά υποδέντρο (αναδρομική κλήση)
 - Αλλιώς (ίδια τιμή), αγνόησε τον κόμβο (υπάρχει ήδη)



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 19** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
[http://media.pearsoncmg.com/ph/esm/deitel/cpp htp 9/code examples/Code Examples.zip](http://media.pearsoncmg.com/ph/esm/deitel/cpp_htp_9/code_examples/Code_Examples.zip)



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 9: C++ ΕΙΣΟΔΟΣ - ΕΞΟΔΟΣ / ΑΛΦΑΡΙΘΜΗΤΙΚΑ / ΑΡΧΕΙΑ

Κανάλια Εισόδου - Εξόδου

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Είσοδος / Έξοδος (Streams)

Βιβλιοθήκη `iostream`

- Βιβλιοθήκη `iostream`
 - Περιέχει εκατοντάδες βιβλιοθήκες για δυνατότητες Εισόδου/Εξόδου
 - `<iostream.h>`
 - Κύρια Είσοδος (`cin`)
 - Κύρια Έξοδος (`cout`)
 - Unbuffered error (`cerr`)
 - Buffered error (`clog`)
 - `<iomanip.h>`
 - Μορφοποιημένη Είσοδος/Έξοδος με stream manipulators
 - `<fstream.h>`
 - Λειτουργίες επεξεργασίας αρχείων



Βιβλιοθήκη `iostream`

- Η βιβλιοθήκη `iostream` διαθέτει templates:
 - `basic_istream` (είσοδος από κανάλι - stream)
 - `basic_ostream` (έξοδος σε κανάλι - stream)
 - `basic_iostream` (είσοδος και έξοδος απο/σε κανάλι)
- Η `typedef` ορίζει ψευδώνυμα για τύπους δεδομένων
 - `typedef Card *CardPtr;`
 - `CardPtr` συνώνυμο με `Card *`
 - `typedefs istream, ostream, iostream`
 - Επιτρέπουν είσοδο/έξοδο χαρακτήρων



Κύρια Κανάλια

- Τελεστές `<<` και `>>`
 - Τελεστές εισόδου και εξόδου από κανάλι (stream)
- **cin**
 - Είναι αντικείμενο **istream**
 - Συνδέεται με την κύρια είσοδο (συνήθως το πληκτρολόγιο)
 - **cin >> var;**
 - Ο Compiler αποφασίζει τον τύπο της μεταβλητής var
 - Καλεί την κατάλληλη υπερφορτωμένη συνάρτηση
- **cout**
 - Είναι αντικείμενο **ostream** object
 - Συνδέεται με την κύρια έξοδο (συνήθως η οθόνη)
 - **cout << var;**
 - Όμοια
- **cerr, clog**
 - Αντικείμενα **ostream**
 - Σύνδεση με το κυρίως κανάλι σφαλμάτων
 - Το **cerr** βγάζει την έξοδο απευθείας
 - Το **clog** κρατάει την έξοδο σε buffer



Επεξεργασία Αρχείων

- Η επεξεργασία αρχείων στην C++ είναι παρόμοια:
 - Class templates
 - `basic_ifstream` (είσοδος από αρχείο)
 - `basic_ofstream` (έξοδος σε αρχείο)
 - `basic_fstream` (είσοδος και έξοδος απο/σε αρχείο)
 - Επιτρέπουν είσοδο/έξοδο χαρακτήρων:
 - `typedefs`
 - `ifstream`
 - `ofstream`
 - `fstream`



Είσοδος / Έξοδος

■ Έξοδος

- Χρήση **ostream**
- Χρήση του τελεστή << για τους βασικούς τύπους
- Χαρακτήρες (χρήση της συνάρτησης **put**)
- Ακέρατοι (δεκαδικοί, οκταδικοί, δεκαεξαδικοί)
- Κινητής Υποδιαστολής
 - Ακρίβεια, δεκαδικά ψηφία, επιστημονική σήμανση
- Justified, padded data
- Έλεγχος κεφαλαίων/πεζών

■ Είσοδος

- Χρήση **istream**
- Χρήση του τελεστή >>
- Επιστρέφει 0 όταν συναντήσει EOF
 - Διαφορετικά επιστρέφει αναφορά σε αντικείμενοj
 - **cin >> var**
- Αν προκύψει σφάλμα ενημερώνονται τα State bits



Είσοδος Χαρακτήρα

- **get** συνάρτηση
 - **cin.get()**
 - Επιστρέφει έναν χαρακτήρα ή EOF
- End-of-file (EOF)
 - σηματοδοτεί το τέλος της εισόδου
 - Ανάλογα το σύστημα: *ctrl-z* ή *ctrl-d*
 - **cin.eof()**
 - Επιστρέφει **1 (true)** αν έχει συμβεί το EOF



Παράδειγμα

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
int main() {
    int character;
    cout << "EOF" << cin.eof() << endl
    while ( ( character = cin.get() ) != EOF )
        cout.put( character );
    cout << "\nEOF in this system is: " << character << endl;
    cout << "EOF" << cin.eof() << endl return 0; }
```

```
cin.EOF:0
test sentence
test sentence
^Z
Last Charachter entered (EOF): -1
cin.EOF::1
```



Είσοδος χαρακτήρα/ων

- **get (charRef)**
 - Ως όρισμα παίρνει χαρακτήρα με αναφορά
 - Ανακτά έναν χαρακτήρα και τον αποθηκεύει στην **charRef**
 - Returns reference to **istream**
 - Επιστρέφει -1, αν διαβάσει EOF
- **get (charArray, size, delimiter)**
 - Διαβάζει έως **size-1** χαρακτήρες, μέχρι να σηναντήσει τον χαρακτήρα **delimiter**
 - Αν δεν δοθεί **delimiter** θεωρείται ο **'\n'**
 - Ο **delimiter** **παραμένει** στο κανάλι εισόδου
 - Μπορεί να αφαιρεθεί με **cin.get()** or **cin.ignore()**
 - Τερματίζει το **charArray** με τον χαρακτήρα τερματισμού **'\0'**
- **getline (array, size, delimiter)**
 - Όμοια με την παραπάνω **get** με τα ίδια ορίσματα
 - **Αφαιρεί** τον χαρακτήρα **delimiter** από το κανάλι εισόδου



Παράδειγμα

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
int main(){
    char buf1[ 60 ];
    char buf2[ 60 ];
    cout<<"Your Sentence:";
    cin >> buf1;
    cout <<"Buf1:" buf1 <<endl;
    cin.get( buf2, 60 );
    cout <<"Buf2:" buf2 ;
}
```

Your Sentence:Soon comes the cold, and the night that never ends.
Buf1:Soon
Buf2: comes the cold, and the night that never ends.



Είσοδος χαρακτήρα/ων

- **ignore ()**
 - Αφαιρεί χαρακτήρες από το κανάλι εισόδου (default 1)
 - Σταματάει όταν συναντήσει delimiter (Default **EOF**)
- **putback ()**
 - Τοποθετεί πίσω στο κανάλι τον χαρακτήρα που ανακτήθηκε με την **get ()**
- **peek ()**
 - Επιστρέφει τον επόμενο χαρακτήρα του καναλιού, χωρίς να τον αφαιρέσει



Error Bits

- Οι τελεστές << και >> είναι υπερφορτωμένοι ώστε να αποδέχονται διάφορους τύπους
- Αν προκύψουν μη αναμενόμενου τύπου δεδομένα τότε ενημερώνονται κατάλληλα **Error Bits** τα οποία μπορούμε να ελέγξουμε για να διαπιστώσουμε αν απέτυχε η εντολή.



Είσοδος / έξοδος Bytes

■ `read (istream)`

- Είσοδος **bytes** σε πίνακα χαρακτήρων
- Αν δεν ανακτηθούν αρκετοί χαρακτήρες ενημερώνεται το **failbit**
- Η **gcount ()** επιστρέφει πόσοι χαρακτήρες ανακτήθηκαν στην τελευταία εντολή εισόδου

■ `write (ostream)`

- Έξοδος bytes από πίνακα χαρακτήρων μέχρι να συναντήσει null



String Manipulators

- Οι παραπονητές καναλιού επιτελούν λειτουργίες μορφοποίησης όπως:
 - Πλάτος (σε χαρακτήρες)
 - Ακρίβεια αριθμών (Precision)
 - Σημαίες Μορφοποίησης
 - Γέμισμα χαρακτήρων
 - Flushing καναλιού
 - Είσοδος νέας γραμμής σε κανάλι εξόδου
 - Αγνόηση κενών από κανάλι εισόδου



Παραποίηση Ακεραίων

- Κανονικά η βάση των ακεραίων θεωρείται το 10
 - Μπορούμε να αλλάξουμε βάση με χρήση των:
 - **hex** (βάση το 16)
 - **oct** (βάση το 8)
 - **dec** (επαναφέρει ως βάση το 10)
 - **cout << hex << myInt**
 - **setbase (newBase)** συνάρτηση
 - Δύνατες τιμές: 8, 10, or 16
 - Η παραποίηση ισχύει μέχρι να την αλλάξουμε ρητά.

```
int number;  
cin >> number;  
cout << number << " in hexadecimal is: " << hex  
    << number << endl;
```



Παραποίηση Δεκαδικών

- Παραποίηση της ακρίβειας αριθμών κινητής υποδιαστολής:
 - Αριθμός ψηφίων μετά την υποδιαστολή
 - **setprecision**
 - Ως όρισμα παίρνει πόσα δεκαδικά ψηφία θα εμφανιστούν
 - **cout << setprecision(5)**
 - **precision** συνάρτηση
 - **cout.precision(newPrecision)**
 - Η παραποίηση ισχύει μέχρι να την αλλάξουμε ρητά.



Παραποίηση Δεκαδικών

- **width** συνάρτηση (κλάσης `ios_base`)
 - `cin.width(5)`
 - Καθορίζει το πλάτος (σε χαρακτήρες)
 - Αριθμός χαρακτήρων στην έξοδο
 - Μέγιστος αριθμός στην είσοδο
 - Εναλλακτικά μπορεί να χρησιμοποιηθεί ο `setw` manipulator (`<< setw`)
 - Όταν διαβάζει πίνακα χαρακτήρων διαβάζει έναν λιγότερο για να τοποθετήσει στο τέλος το null
- **showpoint**
 - Αναγκάζει δεκαδικούς να εκτυπωθούν με συγκεκριμένο αριθμό χαρακτήρων προσθέτοντας μηδενικά στο τέλος (πχ 15.400000 αντί για 15.4)
 - Επαναφορά με την `noshowpoint`



Στοίχιση / Πρόσημο

- **left** manipulator
 - Στοίχιση αριστερά με γέμισμα των θέσεων δεξιά
- **right** manipulator
 - Στοίχιση δεξιά με γέμισμα των θέσεων αριστερά
- **internal**
 - Το πρόσημο στοιχίζεται αριστερά, η τιμή δεξιά
$$\begin{array}{r} + \\ 123 \end{array}$$
- **showpos**
 - υποχρεώνει την εκτύπωση του προσήμου
 - Επαναφορά με **noshowpos**



Χαρακτήρας γεμίσματος

- Ορισμός του χαρακτήρα που θα χρησιμοποιηθεί όταν χρειαστεί να καλυφθούν θέσεις
 - **fill** συνάρτηση

```
cout.fill('*')
```
 - **setfill** manipulator

```
<< setfill( '^' )
```



Κεφαλαία / Πεζά

- **uppercase** manipulator
 - Μετατρέπει χαρακτήρες σε κεφαλαία
 - lowercase η προκαθορισμένη συμπεριφορά
 - Επαναφορά με **nouppercase**



Έλεγχος Σφαλμάτων

- **rdstate()** συνάρτηση
 - Επιστρέφει την κατάσταση σφαλμάτων του καναλιού
 - Ελέγχει για **goodbit**, **badbit**, etc.
 - Συνιστάται η χρήση των: **good()**, **bad()**
- **clear()** συνάρτηση
 - Προκαθορισμένη τιμή ορίσματος: **goodbit**
 - Θέτει την κατάσταση σε "good", ώστε να συνεχίσει η είσοδος/έξοδος
 - Can pass other values
 - **cin.clear(ios::failbit)**
Αλλάζει τιμή στο **failbit**



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 13** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 9: C++ ΕΙΣΟΔΟΣ - ΕΞΟΔΟΣ / ΑΛΦΑΡΙΘΜΗΤΙΚΑ / ΑΡΧΕΙΑ

Διαχείριση Αρχείων

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Διαχείριση Αρχείων

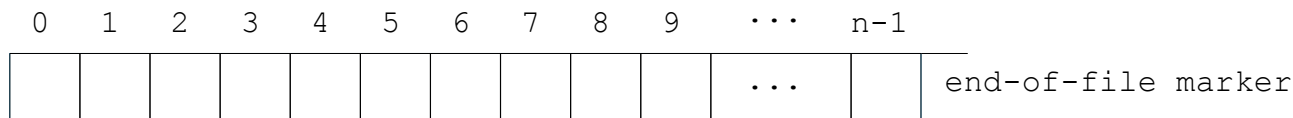
Ιεραρχία Δεδομένων

- Από μικρότερα σε μεγαλύτερα:
 - Bit (δυναδικό ψηφίο 1 ή 0)
 - Byte: 8 bits
 - Μπορεί να αποθηκεύσει έναν χαρακτήρα (**char**)
 - Unicode για μεγάλα σετ χαρακτήρων (**wchar_t**)
 - Πεδίο (Field): γκρουπ χαρακτήρων με κάποιο νόημα (πχ ένα όνομα)
 - Εγγραφή (Record): ομάδα από σχετικά πεδία (**struct** ή **class** στην C++)
 - Αρχείο (File): ομάδα σχετικών εγγραφών
 - Βάση Δεδομένων (Database): ομάδα σχετικών αρχείων



Αρχεία στην C++

- Η C++ χειρίζεται τα αρχεία ως αλληλουχία bytes
 - το τέλος σηματοδοτείται με: *end-of-file*



- Όταν ανοίγει ένα αρχείο
 - Δημιουργείται αντικείμενο και συνδέεται με ένα κανάλι (stream)
 - **cin**, **cout**, κτλ. όταν περιλαμβάνεται η **<iostream>**



Κανάλια - Streams

- Για να διαχειριστούμε ένα αρχείο:
 - Κάνουμε include τις `<iostream>` και `<fstream>`
 - Class templates
 - `basic_ifstream` (είσοδος)
 - `basic_ofstream` (έξοδος)
 - `basic_fstream` (I/O)
 - `typedefs` ειδικά για `char` I/O
 - `ifstream` (είσοδος `char`)
 - `ofstream` (εξοδος `char`)
 - `fstream` (`char` I/O)



Τρόποι ανοίγματος αρχείου

- Τρόποι ανοίγματος αρχείου (File-open modes)

Mode	Description
<code>ios::app</code>	Εγγραφή στο τέλος του αρχείου
<code>ios::ate</code>	Άνοιγμα αρχείου για εγγραφή και μεταφορά στο τέλος του αρχείου. Μπορεί να γίνει εγγραφή σε οποιοδήποτε σημείο του αρχείου.
<code>ios::in</code>	Άνοιγμα αρχείου για είσοδο
<code>ios::out</code>	Άνοιγμα αρχείου για εγγραφή
<code>ios::trunc</code>	Διαγραφή των περιεχόντων του αρχείου αν υπάρχει (προκαθορισμένη συμπεριφορά και του <code>ios::out</code>)
<code>ios::binary</code>	Ανοίγει αρχείο για δυαδική ανάγνωση/εγγραφή (π.χ., όχι κείμενο)

- `ofstream` για έξοδο

- `ofstream outClientFile("persons.dat", ios::out);`
- `ofstream outClientFile("persons.dat");`



Παράδειγμα – Έξοδος

```
#include <iostream>
using std::cout;
using std::cin;
using std::ios;
using std::cerr;
using std::endl;
#include <fstream>
using std::ofstream;
#include <cstdlib>

int main(){
    ofstream outClientFile("persons.dat", ios::out );
    if ( !outClientFile ) {
        cerr << "File could not be opened" << endl;
        return 1;
    }
    char name[30];
    int age;
    while ( cin >> name >> age )
        outClientFile << name << ' ' << age << endl;
    return 0;
}
```



Παράδειγμα – Είσοδος

```
#include <iostream>
using std::cout;
using std::ios;
using std::cerr;
using std::endl;
#include <fstream>
using std::ifstream;
#include <iomanip>
#include <cstdlib>

int main(){
    ifstream inClientFile("persons.dat", ios::in );
    if ( !inClientFile ) {
        cerr << "File could not be opened" << endl;
        return 1;
    }
    char name[ 30 ];
    int age;
    while ( inClientFile >> name >> age )
        cout << name << ' ' << age << endl;
    return 0;
}
```



Δείκτες σε θέση αρχείου

- Δείκτες σε θέση στο αρχείο
 - Number of next byte to read/write
 - Συναρτήσεις αλλαγής θέσης του δείκτη
 - **seekg** (seek get της **istream** class)
 - **seekp** (seek put της **ostream** class)
 - Οι κλάσεις διαθέτουν "get" και "put" δείκτες
 - **seekg** και **seekp** παίρνουν ως ορίσματα:
 - Offset: αριθμός bytes
 - Direction (προκαθορισμένη τιμή **ios::beg**)
 - **ios::beg** – σε σχέση με την αρχή του αρχείου
 - **ios::cur** – σε σχέση με την τρέχουσα θέση
 - **ios::end** – σε σχέση με το τέλος



Άνοιγμα Αρχείου

- Η C++ δεν αναγκάζει συγκεκριμένη δομή στο αρχείο
 - Η έννοια μιας “εγγραφής” καθορίζεται/υλοποιείται από τον προγραμματιστή
- Για να ανοίξει το αρχείο δημιουργείται αντικείμενο
 - δημιουργεί «γραμμή επικοινωνίας» από το αντικείμενο στο αρχείο
 - Κλάσεις
 - `ifstream` (μόνο για είσοδο)
 - `ofstream` (μόνο για έξοδο)
 - `fstream` (I/O)
 - Οι δημιουργοί παίρνουν ως παραμέτρους το όνομα του αρχείου (*file name*) και τον τρόπο (*file-open mode*)

```
ofstream outClientFile( "filename", fileOpenMode );
```
 - Για να ανοίξουμε ένα αρχείο αργότερα (αφού δημιουργήσουμε το αντικείμενο):

```
ofstream outClientFile;  
outClientFile.open( "filename", fileOpenMode);
```



Παραδείγματα seekg

- Παραδείγματα
 - `fileObject.seekg(0)`
 - Πηγαίνει στην αρχή του αρχείου (location 0), προκαθορισμένο:
`ios::beg`
 - `fileObject.seekg(n)`
 - Πηγαίνει στο n-οστό byte από την αρχή
 - `fileObject.seekg(n, ios::cur)`
 - Μεταφέρεται n bytes μπροστά
 - `fileObject.seekg(y, ios::end)`
 - Πηγαίνει πίσω y bytes από το τέλος
 - `fileObject.seekg(0, ios::cur)`
 - Πηγαίνει στο τελευταίο byte
 - `seekp` (όμοια)



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 14** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»





ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά
μαθήματα ΠΠ

Οντοκεντρικός Προγραμματισμός

Ενότητα 9: C++ ΕΙΣΟΔΟΣ - ΕΞΟΔΟΣ / ΑΛΦΑΡΙΘΜΗΤΙΚΑ / ΑΡΧΕΙΑ

Αλφαριθμητικά

ΔΙΔΑΣΚΟΝΤΕΣ: Ιωάννης Χατζηλυγερούδης, Χρήστος Μακρής

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Αλφαριθμητικά

Αρχικοποίηση

- Template class **basic_string**
 - Μεταχείριση αλφαριθμητικών (αντιγραφή, αναζήτηση κτλ.)
 - `typedef basic_string< char > string;`
 - Also `typedef` for `wchar_t`
 - Βιβλιοθήκη: `<string>`
- Αρχικοποίηση **string**
 - `string s1("word");` `word`
 - `string s2(5, 'x');` `xxxxxx`
 - `string s3 = "word"` `word`
 - Έμμεσα καλεί τον δημιουργό



Αρχικοποίηση

- Δεν επιτρέπεται μετατροπή από `int` ή `char`
 - Τα παρακάτω προκαλούν σφάλματα:
 - `string err1 = 'd';`
 - `string err2('x');`
 - `string err3 = 45;`
 - `string err4(5);`
 - Μπορούμε να αναθέσουμε έναν χαρακτήρα με :
 - `s = 'n';`



Χαρακτηριστικά

- **string**

- Δεν είναι απαραίτητο να τερματίζονται με `null`
- Η μέθοδος `length` επιστρέφει τον αριθμό χαρακτήρων:
`s1.length()`
- Ο τελεστής `[]` επιτρέπει προσπέλαση χαρακτήρων: `s1[0]`
- Ένα `string` δεν είναι δείκτης
- Ανάκτηση από κανάλι
 - `cin >> stringObject;`
 - `getline(cin, s)`
 - Διαβάζει μέχρι τον χαρακτήρα επόμενης γραμμής.



Ανάθεση

- Ανάθεση
 - `s2 = s1;` ή `s2.assign(s1);`
 - Δημιουργεί αντίγραφο
 - `myString.assign(s, start, N);`
 - αντιγράφει **N** χαρακτήρες από το string **s**, ξεκινώντας από τη θέση **start**
 - Ανάθεση συγκεκριμένων θέσεων (χαρακτήρων)
 - `s1[0] = s2[4];`



Μέθοδοι `at` και `append`

- Έλεγχος εμβέλειας
 - `s.at(index);`
 - Επιστρέφει τον χαρακτήρα στην θέση `index`
 - Μπορεί να πετάξει εξαίρεση `out_of_range` αν δοθεί αριθμός εκτός ορίων
 - Ο τελεστής `[]` δεν κάνει τέτοιο έλεγχο
- Προσθήκη στο τέλος (Concatenation)
 - `s.append("winter");` ή `s += "winter";`
 - Προσθέτουν το `"pet"` στο τέλος του `s`
 - `s.append(ssrc, start, N);`
 - Τοποθετεί στο τέλος του `s`, `N` χαρακτήρες από τον `ssrc`, ξεκινώντας από την θέση `start`



Σύγκριση

- Υπερφορτωμένοι τελεστές σύγκρισης:
 - `==`, `!=`, `<`, `>`, `<=` και `>=`
 - Επιστρέφουν τιμή `bool`
- `s1.compare(s2)`
 - Επιστρέφει θετικό αν το `s1` είναι μεγαλύτερο λεξικογραφικά
 - Σύγκριση γράμμα προς γράμμα
 - Αρνητικό αν είναι μικρότερο, 0 αν είναι ίσα
 - `s1.compare(start, length, s2, start, length)`
 - Συγκρίνει επιμέρους μέρη του `s1` και `s2`
 - `s1.compare(start, length, s2)`
 - Συγκρίνει επιμέρους μέρος του `s1` με ολόκληρο το `s2`



Μέθοδοι `substr` , `switch`

- Η `substr` επιστρέφει ένα μέρος του αλφαριθμητικού
 - `s.substr(start, N) ;`
 - Επιστρέφει ένα string παίρνοντας `N` χαρακτήρες του `s`, ξεκινώντας από την θέση `start`
- Μέθοδος `switch`
 - `s1.swap(s2) ;`
 - Ανταλλάσσει τα περιεχόμενα των δύο strings



Χρήσιμες Μεθόδους

- **s1.size()** και **s1.length()**
 - Επιστρέφει το μέγεθος του string
- **s1.capacity()**
 - Επιστρέφει πόσα στοιχεία μπορούν να αποθηκευτούν χωρίς να γίνει reallocation
- **s1.max_size()**
 - Επιστρέφει το μέγιστο μέγεθος του string
- **s1.empty()**
 - Επιστρέφει true αν είναι άδειο
- **s1.resize(newlength)**
 - Αλλάζει το μέγεθος σε newlength



Μέθοδοι αναζήτησης

- Επιστρέφουν την θέση που βρέθηκε
- Αν δεν βρεθεί επιστρέφουν, `string::npos` (σταθερά)
 - `s1.find(s2)`
 - `s1.rfind(s2)`
 - Ξεκινάει την αναζήτηση από δεξιά
 - `s1.find_first_of(s2)`
 - Επιστρέφει την πρώτη εμφάνιση στο `s1` οποιουδήποτε χαρακτήρα του `s2`
 - `s1.find_last_of(s2)`
 - Επιστρέφει την τελευταία εμφάνιση στο `s1` οποιουδήποτε χαρακτήρα του `s2`
 - `s1.find_first_not_of(s2)`
 - Επιστρέφει την πρώτη εμφάνιση στο `s1` οποιουδήποτε χαρακτήρα που δεν υπάρχει στο `s2`
 - `s1.find_last_not_of(s2)`
 - Επιστρέφει την τελευταία εμφάνιση στο `s1` οποιουδήποτε χαρακτήρα που δεν υπάρχει στο `s2`



Αφαίρεση - Αντικατάσταση

- **s1.erase(start)**
 - Αφαίρεση από τη θέση **start** μέχρι το τέλος
- **s1.replace(begin, N, s2)**
 - Ξεκινώντας από την θέση **begin** του **s1** αντικατέστησε τους επόμενους **N** χαρακτήρες, με χαρακτήρες από το **s2**
- **s1.replace(begin, N, s2, index, num)**
 - Όμοια με παραπάνω αλλά καθορίζεται επιπλέον από ποιο σημείο **index** του **s2** θα ξεκινήσουμε και πόσους χαρακτήρες θα χρησιμοποιήσουμε
- Η θέση **string::npos** αναπαριστά το μέγιστο μέγεθος του **string**



Ένθεση

- `s1.insert(index, s2)`
 - Ενθέτει το `s2` πριν από τη θέση `index`
- `s1.insert(index, s2, index2, N) ;`
 - Ενθέτει ένα μέρος του `s2` (`N` χαρακτήρες ξεκινώντας από τη θέση `index2`) πριν από τη θέση `index`



Μετατροπή

- Μέθοδοι μετατροπής:
 - `s1.copy(ptr, N, index)`
 - αντιγράφει `N` χαρακτήρες στον πίνακα `ptr` (`*char`)
 - Ξεκινάει από την θέση `index`
 - Πρέπει να τερματιστεί με `null`
 - `s1.c_str()`
 - Επιστρέφει `const char *`
 - Δεν χρειάζεται να τερματιστεί με `null`
 - `s1.data()`
 - Επιστρέφει `const char *`
 - Πρέπει να τερματιστεί με `null`



Iterators

- Iterators
 - Διαπέραση χαρακτήρων (από αρχή ή από το τέλος)
 - Πρόσβαση στους επιμέρους χαρακτήρες
 - Όμοια με δείκτες
- Βασική Χρήση
 - Δημιουργία
 - `string::const_iterator i = s.begin();`
 - `const`, δεν επιτρέπεται τροποποίηση
 - Αναφορά
 - `*i;` // αναφορά στον χαρακτήρα
 - `++i;` // μεταφορά στον επόμενο χαρακτήρα
 - Έλεγχος για τέλος
 - `i != s.end()`
 - `end` επιστρέφει iterator μετά το τελευταίο στοιχείο του `s`



String Streams

- I/O of strings to and from memory
 - Called in-memory I/O or string stream processing
 - Classes
 - `istringstream` (είσοδος από string)
 - `ostringstream` (έξοδος σε string)
 - `<sstream>` `<iostream>`
- Κανάλι εξόδου `ostringstream`
 - `ostringstream outputString;` Δημιουργία string-καναλιού
 - `outputString << s1 << s2;` Αποθήκευση strings στο string-κανάλι
 - Μέθοδος: `str`
 - Επιστρέφει ως `string` το περιεχόμενο του string-καναλιού
 - `outputString.str()`
- Κανάλι εισόδου `istringstream`
 - `istringstream inputString (myString);` Δημιουργία string-καναλιού από string
 - `inputString >> string1 >> string2` Ανάκτηση από το string-κανάλι σε strings
 - Με παρόμοιο τρόπο που διαβάζουμε από την κύρια είσοδο, `cin`



Πρόσθετο Υλικό

- Μελετήστε και τα παραδείγματα από το **Κεφάλαιο 21** του βιβλίου:
«C++ How to Program, 9/e Paul & Harvey Deitel»
http://media.pearsoncmg.com/ph/esm/deitel/cpp_hpt_9/code_examples/Code_Examples.zip



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.1



Σημείωμα Αναφοράς

Copyright: Πανεπιστήμιον Πατρών, Ιωάννης Χατζηλυγερούδης, 2015.
«Οντοκεντρικός Προγραμματισμός». Έκδοση: 1.0.1 Πάτρα 2015. Διαθέσιμο
από τη δικτυακή διεύθυνση:

<https://eclass.upatras.gr/courses/CEID1105/>



Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.



Σημείωμα Χρήσης Έργων Τρίτων

- Οι διαφάνειες βασίζονται στο βιβλίο «C++ How to Program, 8th Edition, Harvey M. Deitel, Paul J. Deitel, Prentice Hall.»

