

**Ανοικτά Ακαδημαϊκά Μαθήματα**

**Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας**

**Αντικειμενοστρεφής Προγραμματισμός (Ε)**

**Ενότητα 5:** Δείκτες και διαχείριση μνήμης

Κλειώ Σγουροπούλου

Τμήμα Μηχανικών Πληροφορικής ΤΕ

|  |  |
| --- | --- |
| Το περιεχόμενο του μαθήματος διατίθεται με άδεια Creative Commons εκτός και αν αναφέρεται διαφορετικά | 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. |

Πίνακας Περιεχομένων

[Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 5 1](#_Toc402878027)

[Δείκτες και διαχείριση μνήμης (Pointers and memory management) 1](#_Toc402878028)

[Δέσμευση χώρου 1](#_Toc402878029)

[Μια κλάση με δυναμική δέσμευση μνήμης 2](#_Toc402878030)

[1. Υλοποίηση constructors – destructor 2](#_Toc402878031)

[2. Copy constructor 4](#_Toc402878032)

[Assignment operator 8](#_Toc402878033)

[Παρατηρήσεις 13](#_Toc402878034)

[Παράρτημα – Ο πλήρης κώδικας της myString 13](#_Toc402878035)

# Αντικειμενοστρεφής Προγραμματισμός- Εργαστήριο 5

## Δείκτες και διαχείριση μνήμης (Pointers and memory management)

Ως τώρα, στα προγράμματα που είδαμε καθώς και στις εργασίες που δόθηκαν στα πλαίσια του μαθήματος χρησιμοποιήθηκαν έτοιμες κλάσεις/βιβλιοθήκες της C++, οι οποίες διευκόλυναν πολύ το έργο μας, καθώς μας προστάτευσαν από τα πιο δύσκολα θέματα της γλώσσας, όπως είναι η δυναμική δέσμευση μνήμης και οι pointers.

Πράγματι, χρησιμοποιώντας την κλάση string για τις συμβολοσειρές μας, δεν χρειάστηκε να ασχοληθούμε με τα C strings (pointers σε char, δηλαδή char\*), η χρήση των οποίων απαιτεί μεγαλύτερη προσοχή από μέρους μας, καθώς εδώ πρέπει να δεσμεύσουμε εμείς τον απαραίτητο χώρο μνήμης και να προσέξουμε κατά την αντιγραφή να μην ξεπεράσουμε τα όρια του χώρου αυτού. Αντίθετα, η κλάση string ενθυλακώνει όλη αυτή την πολυπλοκότητα, παρέχοντας προς τους χρήστες (εμάς δηλαδή) ένα απλοποιημένο και ασφαλές τρόπο χρήσης (όπως ακριβώς θα πρέπει να λειτουργούν και οι δικές μας κλάσεις)

Επίσης, η κλάση vector που επίσης χρησιμοποιούμε ως τώρα κρύβει μέσα της ακόμα μεγαλύτερη πολυπλοκότητα. Αν εμείς θα θέλαμε να φτιάξουμε μια κλάση αποθήκευσης αντικειμένων, θα πρέπει να προσέξουμε πολλά πράγματα, κυρίως όσον αφορά την κατάλληλη δέσμευση χώρου, αλλά και την αντιγραφή των αντικειμένων.

Σε αυτό και στο επόμενο εργαστήριο θα δούμε πως υλοποιούνται εσωτερικά οι παραπάνω μηχανισμοί, ώστε να είμαστε σε θέση να καταλάβουμε πλήρως πως δουλεύει η C++.

## Δέσμευση χώρου

Για να δεσμεύσουμε δυναμικά μνήμη στην C++ χρησιμοποιούμε τον τελεστή new:

char\* str = new char[10]; //δέσμευση 10 bytes

Κάθε κομμάτι μνήμης που δεσμεύεται θα πρέπει πάντα να ελευθερώνεται όταν παύει να μας είναι χρήσιμο. Για την διαγραφή/απελευθέρωση της μνήμης αυτής χρησιμοποιούμε τον τελεστή delete:

delete []str; //προσοχή στις αγκύλες!

Μπορούμε να χρησιμοποιούμε pointers σε objects και να δεσμεύσουμε χώρο για ένα νέο αντικείμενο:

Point \*obj; //pointer σε MyClass

obj = new Point(2, 3); //δεσμεύουμε χώρο και καλούμε παράλληλα τον

constructor

Για να αποκτήσουμε πρόσβαση στα περιεχόμενα του αντικειμένου (στα public βέβαια) χρησιμοποιούμε τον τελεστή ->

obj->x = 2;

obj->y = 4;

Το αντικείμενο αυτό θα πρέπει να καταστραφεί ως εξής:

delete obj; //χωρίς αγκύλες

## Μια κλάση με δυναμική δέσμευση μνήμης

Στο κεφάλαιο αυτό θα δούμε βήμα-βήμα μια υποθετική υλοποίηση της κλάσης string, την οποία θα ονομάσουμε myString. Η υλοποίηση δεν είναι όσο πολύπλοκη είναι αυτή της κλάσης string, καθώς σκοπός της είναι η κατανόηση των βασικών νοημάτων.

### Υλοποίηση constructors – destructor

#include <string>

class myString

{

char \*str;

public:

myString()

{

str = new char[1];

strcpy(str, "");

}

myString(char \*s)

{

int length = strlen(s);

str = new char[length + 1]; // +1? για τον χαρακτήρα '\0'

strcpy(str, s);

}

~myString()

{

delete []str;

}

};

int main()

{

char\* arr = new char[6];

strcpy(arr, "hello");

myString s(arr); myString ss;

}

Η κλάση myString περιέχει έναν pointer σε char, ένα string της C δηλαδή. Αυτό που πρέπει να προσέξουμε είναι ότι ένα αντικείμενο της κλάσης myString θα περιέχει μόνο τον pointer, ενώ το string στο οποίο δείχνει αυτός ο pointer είναι αποθηκευμένο σε μνήμη εκτός του αντικειμένου. Ένα αντικείμενο της κλάσης myString φαίνεται στο παρακάτω σχήμα:

Προχωράμε στον constructor myString(char \*s), ο οποίος λαμβάνει ως παράμετρο ένα C string (στην περίπτωσή μας το “hello”). Ένα πολύ συχνό λάθος που γίνεται είναι η απευθείας ανάθεση του string αυτού στον pointer str, δηλαδή:

str = s; //λάθος

Αυτό που κάνουμε εδώ είναι αντιγραφή δεικτών και όχι αντιγραφή συμβολοσειρών, δηλαδή οι δύο δείκτες δείχνουν στην ίδια συμβολοσειρά. Αυτό μπορεί να μην δημιουργήσει άμεσα πρόβλημα, αλλά μακροπρόθεσμα είναι πιθανή αιτία λάθους, καθώς το string το οποίο εμείς αναθέσαμε στο str μπορεί να σβηστεί (να γίνει delete) από κάποιον άλλον. Το αποτέλεσμα αυτού θα είναι ο δείκτης str να δείχνει σε θέσεις μνήμης που νομίζει ότι του ανήκουν, αλλά στην πραγματικότητα δεν του ανήκουν.

s

str

Αν κάνουμε delete [ ]arr, τότε ο pointer str θα δείχνει σε απελευθερωμένη μνήμη

H e l l o \0

arr

Αυτό που πρέπει να γίνει εδώ είναι:

* 1. δέσμευση εκ νέου χώρου μνήμης για τον δείκτη str, και
	2. αντιγραφή (με strcpy) του ενός string στο άλλο.

Έτσι ο δείκτης str είναι τελείως αυτόνομος και ανεπηρρέαστος σε αλλαγές που θα γίνουν στον άλλον.



|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |

Τώρα, αν κάνουμε delete[ ] arr, τότε το string του αντικειμένου μας θα μείνει ανεπηρρέαστο.


#### **Άσκηση 1**

Να ορίσετε μια κλάση Person η οποία να ορίζει τις εξής παραμέτρους (να χρησιμοποιήσετε

pointers για τα strings):

* Όνομα
* Επώνυμο
* Ηλικία
1. Να υλοποιήσετε τους 2 απαραίτητους constructors, τον destructor της κλάσης, τις απαραίτητες setter συναρτήσεις (π.χ. setName), καθώς και μια συνάρτηση display.
2. Στην main, να δημιουργήσετε ένα αντικείμενο της κλάσης Person και να καλέσετε την display, ώστε να εκτυπωθούν στην οθόνη τα στοιχεία του ατόμου.
3. Στην main, να ορίσετε ένα πίνακα που θα περιέχει 10 αντικείμενα τύπου Person, να δώσετε τιμές στα πρώτα 2 αντικείμενα και να καλέσετε την συνάρτηση display για τα πρώτα δύο αντικείμενα.

Φροντίστε να βάλετε μηνύματα μέσα σε κάθε συνάρτηση, ώστε να γίνεται φανερό πότε καλείται κάθε συνάρτηση.

### Copy constructor

Το πρόβλημα με τους pointers και την δυναμική δέσμευση μνήμης είναι το ότι η μνήμη αυτή

δεν βρίσκεται μέσα στο αντικείμενο αλλά σε άλλο χώρο (στον σωρό heap).

Αν δοκιμάσουμε να περάσουμε ένα αντικείμενο τύπου myString ως παράμετρο σε μια συνάρτηση (pass by value, σε αντίθεση με το pass by reference), τότε θα φτιαχτεί ένα αντίγραφο του αντικειμένου αυτού. Θυμηθείτε ότι μπορούμε να περάσουμε το ίδιο το αντικείμενο ως παράμετρο (pass by reference) με δύο τρόπους:

* + Περνώντας την αναφορά του (reference)
	+ Περνώντας έναν pointer στο αντικείμενο

Αν δεν κάνουμε κάτι από τα δύο, τότε αυτό που περνάμε στην συνάρτηση είναι το αντίγραφο του αντικειμένου. Το αντίγραφο αυτό φτιάχνεται (γίνεται constructed) από ένα άλλο constructor της κλάσης, τον **copy constructor**.

Η δουλειά του copy constructor ειναι:

1. Να δεσμεύει χώρο για το νέο αντικείμενο, και
2. να αντιγράφει ένα προς ένα τα περιεχόμενα από το αρχικό αντικείμενο στο αντίγραφό του.

Ο copy constructor μας παρέχεται αυτόματα από την C++, δηλαδή δεν είναι απαραίτητο να τον φτιάξουμε εμείς (όπως γίνεται και με τον default constructor) και ονομάζεται default

copy constructor. **Τις περισσότερες περιπτώσεις δεν χρειάζεται να τον υλοποιήσουμε, εκτός από ιδιαίτερες περιπτώσεις, όπως είναι σε κλάσεις που χρησιμοποιούν δυναμική δέσμευση μνήμης, όπως η myString.**

Ας δούμε το εξής παράδειγμα. Προσπαθήστε να εκτελέσετε το παρακάτω πρόγραμμα στον υπολογιστή σας και να καταγράψετε το αποτέλεσμα. Ανάλογα με την πλατφόρμα σας, το πρόγραμμα ενδέχεται να μην ολοκληρωθεί ομαλά.

#include <iostream> using namespace std; class myString

{

public:

char \*str; // την κάναμε προσωρινά public για ευκολία

myString()

{

str = new char[1]; strcpy(str, "");

}

myString(char \*s)

{

int length = strlen(s);

str = new char[length + 1]; // +1? για τον χαρακτήρα '\0'

strcpy(str, s);

}

~myString()

{

delete []str;

}

};

void printmyString(myString ms)

{

//το αντικείμενο ms είναι αντίγραφο

cout << ms.str << endl;

//στο τέλος της συναρτησης θα καταστραφει

}

int main()

{

myString a("hello");

strcpy(a.str, "HELLO"); //προσέχουμε να μην ξεπεράσουμε τα όρια

//εδω περνάμε αντίγραφο του αντικειμένου a

printmyString(a);

//ας προσπαθήσουμε να χειριστούμε πάλι τον πίνακα

//πρόβλημα: ο πίνακας str έχει καταστραφεί!

strcpy(a.str, "hhhhh");

system("pause");

return 0;

}

Στο παραπάνω παράδειγμα ο default copy constructor θα δημιουργήσει μόνο τον pointer str και δεν θα δεσμεύσει χώρο και για την μνήμη όπου αυτός δείχνει. Επίσης, το μόνο που θα κάνει είναι να αντιγράψει τον ενα pointer στον άλλον, δηλαδή τις δεκαεξαδικές τιμές τις οποίες περιέχουν οι δείκτες και αποτελούν ουσιαστικά διευθύνσεις μνήμης.

a

str

H e l l o \0

Λειτουργία του Default copy constructor:

ms

Το αντικείμενο ms είναι το

αντίγραφο του αντικειμένου a που δημιουργείται μέσα στην συνάρτηση printMyString

str

Αντιγράφει μόνο τους pointers Αποτέλεσμα: και οι δύο pointers δείχνουν στο ίδιο string

Αυτή η διαδικασία λέγεται ρηχή αντιγραφή (shallow copy). Το αποτέλεσμα αυτής της ρηχής αντιγραφής θα είναι ότι και οι δύο δείκτες δείχνουν στον ίδιο χώρο μνήμης. Έτσι, όταν κληθεί ο destructor για μια εξ αυτών και απελευθερώσει την μνήμη αυτή, ο άλλος δείκτης θα μείνει να δείχνει σε ανύπαρκτη μνήμη, νομίζοντας ότι του ανήκει. Αν τώρα πάμε μέσω ου δείκτη αυτού να γράψουμε σε αυτή την μνήμη, το πιο πιθανό είναι ότι το πρόγραμμα θα

«σκάσει». Ακόμα και αν δεν τερματιστεί άμεσα, έχει γίνει ζημιά (corrupted memory) η οποία θα εκδηλωθεί σε μεταγενέστερο στάδιο (και δεν θα ξέρουμε τότε ποιά είναι τα αίτια του προβλήματος).

Το σωστό είναι κάθε αντικείμενο να έχει τον δικό του χώρο μνήμης. Για τον λόγο αυτό είναι απαραίτητο να υπερφορτώσουμε τον copy constructor ώστε δημιουργήσουμε το αντίγραφο ενός αντικειμένου όπως εμείς το θέλουμε. Αυτό που πρεπει να κανει ο copy constructor ειναι αντιγραφήσεβάθος(deepcopy).

Αυτός ο copy constructor θα δεσμεύει τον απαραίτητο χώρο για τους pointers (π.χ. τα strings) και μόνο τότε θα κάνει τις απαραίτητες αντιγραφές.

Η σύνταξη ενός copy constructor είναι παρόμοια με αυτή ενός κατασκευαστή. Έχει το ίδιο όνομα με το όνομα της κλάσης, δεν έχει επιστρεφόμενη τιμή και πάντα δέχεται μια const παράμετρο που αποτελεί αναφορά στο αρχικό αντικείμενο.

myString(const myString &s)

{

int length = strlen(s.str);

str = new char[length + 1]; strcpy(str, s.str);

}

Σε αυτό το σημείο, έχουμε γλυτώσει τα προβλήματα που προκύπτουν όταν περνάμε αντικείμενα ως παραμέτρους. Ένας άλλο τρόπος είναι να περάσουμε ως παράμετρο το reference του αντικειμένου ή έναν pointer σε αυτό το αντικείμενο (πέρασμα κατά αναφορά, pass by reference). Αυτή η μέθοδος είναι πιο γρήγορη, καθώς δεν έχουμε κλήση της συνάρτηση του copy constructor (οι συναρτήσεις είναι «ακριβό πράγμα» όταν έχουμε μεγάλες απαιτήσεις σε ταχύτητα). Από την άλλη θα πρέπει να τις χειριζόμαστε με προσοχή.

Η κλάση μας myString δεν είναι ακόμα πλήρης. Αν αποπειραθούμε να αντιγράψουμε ένα string της κλάσης myString σε ένα άλλο θα έχουμε πρόβλημα. Παρακάτω θα ολοκληρώσουμε την κλάση myString ώστε να είναι ασφαλής από κάθε άποψη.

#### **Άσκηση 2**

Να ορίσετε την ακόλουθη συνάρτηση:

void calculateBirthYear(Person p)

{

int currentYear = 2012;

int birthYear = currentYear – p.age; return birthYear;

}

Η συνάρτηση αυτή ΔΕΝ αποτελεί μέλος της κλάσης, αλλά είναι ανεξάρτητη από αυτήν (δεν ανήκει σε καμία κλάση). Για τον λόγο αυτό παίρνει ως παράμετρο ένα αντικείμενο τύπου Person.

1. Εξηγήστε γιατί θα εμφανιστεί πρόβλημα με το πρόγραμμά μας, δεδομένου του τρόπου που έχουμε υλοποιήσει ως τώρα της κλάση Person (προσέξτε τα μηνύματα που εμφανίζονται από κάθε συνάρτηση).
2. Να προτείνετε τρόπους αντιμετώπισης του προβλήματος.
3. Να υλοποιήσετε τον copy constructor της Person.

## Assignment operator

Όπως επισημάνθηκε προηγουμένως, η κλάση μας σε αυτό το σημείο δεν είναι πλήρης (όσον αφορά την ασφάλεια) καθώς θα έχουμε μεγάλο πρόβλημα και κατά την ανάθεση ενός αντικειμένου σε ένα άλλο, όπως φαίνεται στο παρακάτω παράδειγμα:

#include <iostream> using namespace std; class myString

{

public:

char \*str;

myString()

{

str = new char[1];

strcpy(str, "");

}

myString(char \*s)

{

int length = strlen(s);

str = new char[length + 1]; // +1? για τον χαρακτήρα '\0'

strcpy(str, s);

}

~myString()

{

delete []str;

}

myString(myString &s)

{

int length = strlen(s.str); str = new char[length + 1]; strcpy(str, s.str);

}

};

int main()

{

myString a("hello");

strcpy(a.str, "HELLO"); //προσέχουμε να μην ξεπεράσουμε τα όρια

{

//αυτό είναι ένα block εντολών

//Το αντικείμενο b ορίζεται μόνο μέσα σε αυτό

//Μετά το πέρας του block το αντικείμενο b θα καταστραφεί

myString b;

b = a;

}

//ας προσπαθήσουμε να χειριστούμε πάλι τον πίνακα

//πρόβλημα: ο πίνακας str έχει καταστραφεί!

strcpy(a.str, "hhhhh");

system("pause");

return 0;

}

Το πρόβλημα εδώ έγκειται στο ότι η αντιγραφή του αντικειμένου γίνεται πάλι ρηχή, δηλαδή αντιγράφεται μόνο ο δείκτης και όχι και το περιεχόμενο του (εκεί δηλαδή που δείχνει ο δείκτης).

a

str

H e l l o \0

Ανάθεση ενός αντικειμένου σε ένα άλλο σημαίνει αντιγραφή των περιεχομένων του ένα- προς-ένα

Ο Default assignment operator αντιγράφει μόνο τους Pointers. Αποτέλεσμα: Οι δύο Pointers δείχνουν στο ίδιο string

b

str

Μόλις βγούμε από το block κώδικα που ορίσαμε με τις αγκύλες (δηλαδή αμέσως μετά την εντολή b=a;) τότε το αντικείμενο b καταστρέφεται (ορίζεται μόνο μέσα στο block αυτό), οπότε πάλι ο pointer του αντικειμένου a δείχνει σε χώρο μνήμης που δεν του ανήκει:

Το πρόβλημα αυτό λύνεται με την υπερφόρτωση και του τελεστή ανάθεσης (assignment operator) που είδαμε στο Εργαστήριο 4.

myString& operator=(myString &s)

{

if (str!=NULL)

delete []str;

int length = strlen(s.str);

str = new char[length + 1]; strcpy(str, s.str);

return \*this;

}

Η υλοποίηση του τελεστή είναι η ίδια σχεδόν με αυτή του copy constructor, με μόνη διαφορά το ότι πρέπει πρώτα να απελευθερώσουμε την μνήμη στην οποία ήδη δείχνει ο δείκτης str και να δεσμεύσουμε εκ νέου μνήμη για τον δείκτη αυτό, ανάλογα με τις ανάγκες σε χώρο που προκύπτουν.

Αν δεν είχαμε υπερφορτώσει τον τελεστή ανάθεσης, τότε θα έπρεπε να βρούμε ένα «πλάγιο» τρόπο για να κάνουμε την αντιγραφή, πχ. Με μια συνάρτηση copy()

void copy(myString &s)

{

if (str!=NULL)

delete []str;

int length = strlen(s.str); str = new char[length + 1]; strcpy(str, s.str);

}

η οποία θα έπρεπε να κληθεί ως εξής: myString s("hello"); myString s1;

s1.copy(s); // αντι για s1 = s;

Γενικώς ως κανόνα θα πρέπει να θυμάστε να υλοποιήσετε μόνοι σας για κάθε κλάση που εμπεριέχει δυναμική δέσμευση μνήμης:

1. **τους constructors**
2. **τον copy constructor**
3. **τον assignment operator**

**Επίσης δεν θα πρέπει να ξεχάσετε να ελευθερώσετε την μνήμη αυτή στον destructor της κλάσης.**

Στο παράρτημα παρατίθεται μια πιο πλήρης υλοποίηση της κλάσης myString όπου έχουν υπερφορτωθεί και οι τελεστές << και >> ώστε να μπορούμε να εκτυπώνουμε και να διαβάζουμε απευθείας τα αντικείμενα τύπου myString.

####  **Άσκηση 3**

Να υλοποιήσετε τον assignment operator της κλάσης Person

#### **Παράδειγμα υλοποίησης vector**

Ας δούμε τώρα ένα απλό δείγμα υλοποίησης ενός vector. Καθώς ακόμα δεν έχουμε μάθει την χρήση των templates, η κλάση που θα φτιάξουμε θα αφορά ένα vector που μπορεί να περιέχει μόνο αντικείμενα της κλάσης myString

#include <iostream>

#define DEFAULT\_SIZE 10

using namespace std;

//εννοείται ότι κάπου εδώ ορίζεται η κλάση myString

//ή ότι ορίζεται σε ένα άλλο header file

class myVector

{

myString \*vec;

int size;

int nextEmpty;

public:

myVector()

{

//εδώ καλείται ο default constructor της myString 10 φορές

vec = new myString[DEFAULT\_SIZE]; size = DEFAULT\_SIZE;

nextEmpty = 0;

}

myVector(int \_size)

{

//εδώ καλείται ο default constructor της myString \_size φορές

vec = new myString[\_size]; size = \_size;

nextEmpty = 0;

}

~myVector()

{

delete []vec;

}

myVector(myVector &s)

{

//εδώ καλείται ο default constructor της myString 10 φορές

vec = new myString[s.size]; size = s.size;

nextEmpty = s.nextEmpty;

for(int i=0; i<s.nextEmpty; i++)

{

//αυτό θα δουλέψει σωστά μόνο αν η κλάση myString

//έχει υπερφορτώσει τον assignment operator =

vec[i] = s.vec[i];

//αν όχι, τότε μπορούμε να κάνουμε αυτό:

//vec[i].copy(s.vec[i]);

}

}

void push\_pack(myString ms)

{

if(nextEmpty < size)

{

}

else

{

//αυτό είναι σωστό μόνο όταν υπερφορτώσουμε

//τον τελεστή ανάθεσης για την myString

vec[nextEmpty++] = ms;

//ο vector μπορει να δεσμεύσει και άλλο χώρο

//ώστε να μεγαλώσει

//φτιάχνουμε ένα νέο πίνακα μεγαλύτερο κατά 10 θέσεις

myString \*new\_vec = new myString[size + 10];

//αντιγράφουμε σε αυτόν τον υπάρχοντα πίνακά μας

for(int i=0; i<size; i++) new\_vec[i] = vec[i];

//ορίζουμε ένα προσωρινό pointer που

//θα δείχνει στον παλιό πίνακα

//και κάνουμε τον vec να δείχνει στον νέο πίνακα

myString \*temp = vec;

vec = new\_vec;

//Τέλος ελευθερώνουμε την μνήμη του αρχικού πίνακα

//Έχουμε πρόσβαση σε αυτή μέσω του temp pointer!!

delete []temp;

//τέλος, κάνουμε και την προσθήκη

vec[nextEmpty++] = ms;

}

}

int length()

{

return size;

}

};

int main()

{

myVector ss;

//να πως δημιουργούμε νέο αντικείμενο με new

myString \*my;

my = new myString("first");

//H push\_back περιμένει αντικείμενο, όχι pointer,

//άρα θα δώσουμε παράμετρο το \*my

ss.push\_pack(\*my);

//εδώ καλείται ο operator=(char\*) της myString

//Προσέξτε ότι αλλάζουμε το ίδιο αντικείμενο (\*my)

//Δεν υπάρχει πρόβλημα γιατί στην push\_back περναμε αντίγραφο.

\*my = "second";

//Αν περνάγαμε reference θα είχαμε πρόβλημα

//και θα έπρεπε να φτιάξουμε νέο αντικείμενο,

//όχι απλά να αλλάξουμε το παλιό (\*my)

ss.push\_pack(\*my);

system("pause");

return 0;

}

Σε επόμενο μάθημα θα δούμε πως μπορούμε να φτιάξουμε αυτή την κλάση έτσι ώστε να μπορεί να περιέχει αντικείμενα οποιασδήποτε κλάσης, όχι μόνο της myString.

## Παρατηρήσεις

#### **Arrays of objects**

Ένα σημείο που πρέπει να προσεχθεί είναι η δημιουργία πινάκων, μέσα στους οποίους αποθηκεύουμε αντικείμενα. Κατά την δημιουργία του πίνακα, δημιουργούνται και τα αντικείμενα που αυτός περιέχει, δηλαδή καλείται ο default constructor τους.

Σε στατικά ορισμένους πίνακες η κλήση των κατασκευαστών γίνεται κατά την δημιουργία του αντικείμενου που τον περιέχει (στον κατασκευαστή δηλαδή). Σε δυναμικά ορισμένους πίνακες (μέσω δεικτών) η κλήση αυτή γίνεται κατά την δέσμευση μνήμης μέσω του τελεστή new.

#### **Array of pointers to objects**

Αντί να φτιάξουμε ένα πίνακα που θα περιέχει αντικείμενα, έχουμε την επιλογή να φτιάξουμε ένα πίνακα που θα περιέχει pointers σε τέτοια αντικείμενα. Παράδειγμα:

//πίνακας μεταβλητού μεγέθους με δείκτες

myString \*\*vec;

//αρχικοποίηση πίνακα που περιέχει δείκτες

vec = new myString\*[10]; myString my("hello");

//ανάθεση δείκτη υπάρχοντος αντικειμένου

vec[0] = &my;

//δημιουργία νέου αντικειμένου

//και απευθείας ανάθεση του δείκτη του

vec[1] = new myString("world");

## Παράρτημα – Ο πλήρης κώδικας της myString

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

class myString

{

char \*str;

public:

myString()

{

str = new char[1];

strcpy(str, "");

}

myString(char \*s)

{

int length = strlen(s);

str = new char[length + 1]; // +1? για τον χαρακτήρα '\0'

strcpy(str, s);

}

~myString()

{

delete []str;

}

myString(myString &s)

{

int length = strlen(s.str);

str = new char[length + 1]; strcpy(str, s.str);

}

myString& operator=(myString &s)

{

if (str!=NULL)

delete []str;

int length = strlen(s.str); str = new char[length + 1]; strcpy(str, s.str);

return \*this;

}

void copy(myString &s)

{

if (str!=NULL)

delete []str;

int length = strlen(s.str); str = new char[length + 1]; strcpy(str, s.str);

}

myString& operator=(char \*s)

{

if (str!=NULL)

delete []str;

int length = strlen(s);

str = new char[length + 1];

strcpy(str, s);

return \*this;

}

myString operator+(myString s)

{

int length1 = strlen(str);

int length2 = strlen(s.str);

char\* temp\_str = new char[length1 + length2 + 1]; strcpy(temp\_str, str);

strcat(temp\_str, s.str); myString temp(temp\_str);

return temp;

}

int length()

{

return strlen(str);

}

friend ostream& operator<<(ostream&, const myString&);

friend istream & operator>>(istream&, myString&);

};

ostream& operator<<(ostream& output, const myString& s)

{

output << s.str;

return output; // for multiple << operators.

}

istream & operator >> (istream &input, myString &s)

{

char input\_string[200];

input >> input\_string;

if (s.str!=NULL)

delete []s.str;

s.str = new char[strlen(input\_string)+1]; strcpy(s.str, input\_string);

return input;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

myString s("hello");

myString ss; myString s1(s);

ss.copy(s1);

myString ssss = s + s1; cout << ssss << endl;

cin >> ss;

cout << ss << endl;

system("pause");

return 0;

}

|  |
| --- |
| **Ανοικτά Ακαδημαϊκά Μαθήματα****Τεχνολογικό Εκπαιδευτικό Ίδρυμα Αθήνας** |
| **Τέλος Ενότητας** |
| **Χρηματοδότηση*** Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Αθήνας**» έχει χρηματοδοτήσει μόνο τη 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.

 |

**Σημειώματα**

**Σημείωμα Αναφοράς**

Copyright ΤΕΙ Αθήνας, Κλειώ Σγουροπούλου, 2014. Κλειώ Σγουροπούλου. «Αντικειμενοστρεφής Προγραμματισμός». Ενότητα 5: «Δείκτες και διαχείριση μνήμης». Έκδοση: 1.0. Αθήνα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση: [ocp.teiath.gr](https://ocp.teiath.gr/).

**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.