ΣΤΟΙΧΕΙΑ ΤΗΣ ΓΛΩΣΣΑΣ C++

Πέρασμα παραμέτρων, συναρτήσεις δόμησης και αποδόμησης

ΑΝΑΚΕΦΑΛΑΙΩΣΗ

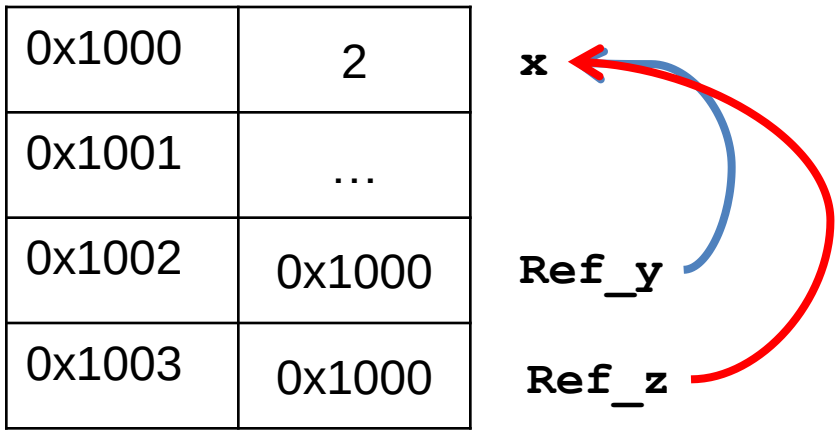
Αναφορές

Όλα αυτά είναι ισοδύναμα

```
int main()
{
    int x = 2;
    int &y = x;
    int &z = y;
}
```

```
int main()
{
    int x = 2;
    int const *Ref_y = &x;
    int const *Ref_z = &(*Ref_y);
}
```

```
int main()
{
    int x = 2;
    int &y = x;
    int &z = x;
}
```



Πέρασμα παραμέτρων

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

int main()
{
   char x[100];
   strcpy(x, "this");
   char y[100];
   strcpy(y , "that");
   cout << x << " " << y << endl;
   // swap the strings
   cout << x << " " << y << endl;
}
```

Πως θα υλοποιήσουμε την swap?

Πέρασμα παραμέτρων

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

int main()
{
  char *x = new char[100];
  strcpy(x, "this");
  char *y = new char[100];
  strcpy(y , "that");
  cout << x << " " << y << endl;
  mySwap(x,y);
  cout << x << " " << y << endl;
}
```

```
void mySwap(char *x, char *y)
{
    char z[100];
    strcpy(z,y);
    strcpy(y,x);
    strcpy(x,z);
}
```

Πέρασμα παραμέτρων δια αναφοράς με δείκτη

Πέρασμα παραμέτρων

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

int main()
{
   int x = 5;
   int y = 10;
   cout << x << " " << y << endl;
   mySwap(x,y);
   cout << x << " " << y << endl;
}
```

```
void mySwap(int &x, int &y)
{
    int z = x;
    x = y;
    y = z;
}
```

Πέρασμα παραμέτρων
δια αναφοράς με αναφορά

Πέρασμα παραμέτρων

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

int main()
{
   char *x = new char[100];
   strcpy(x, "this");
   char *y = new char[100];
   strcpy(y , "that");
   cout << x << " " << y << endl;
   mySwap(x,y);
   cout << x << " " << y << endl;
}
```

```
void mySwap(char *x, char *y)
{
   char *z;
   z = x;
   x = y;
   y = z;
   cout <<x << " " << y << endl;
}
```

Τι έξοδο θα πάρουμε?

```
this that
that this
this that
```

Ο δείκτης περνιέται δια τιμής!

Πέρασμα παραμέτρων

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

int main()
{
   char *x = new char[100];
   strcpy(x, "this");
   char *y = new char[100];
   strcpy(y , "that");
   cout << x << " " << y << endl;
   mySwap(x,y);
   cout << x << " " << y << endl;
}
```

```
void mySwap(char *&x, char *&y)
{
   char *z;
   z = x;
   x = y;
   y = z;
   cout <<x << " " << y << endl;
}
```

Πέρασμα παραμέτρων
δια αναφοράς με αναφορά

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

class myString
{
private:
  char s[100];
public:
  char *GetString();
  void SetString(char const *);
  // Swap ??
};

char * myString::GetString()
{
  return s;
}

void myString::SetString(char const *sNew)
{
  strcpy(s,sNew); //missing checks!
}

int main(){
  myString mS1, mS2;
  mS1.SetString("this");
  mS2.SetString("that");
  cout << mS1.GetString() << " " << mS2.GetString() << endl;
}
```

```
class myString
{
private:
  char s[100];
public:
  char *GetString();
  void SetString(char const *);
  void Swap(myString &);
};
```

```
void myString::Swap(myString &other)
{
  char t[100];
  char *o= other.GetString();
  strcpy(t,o);
  other.SetString(s);
  strcpy(s,t);
}
```

```
int main(){
  myString mS1, mS2;
  mS1.SetString("this");
  mS2.SetString("that");
  cout << mS1.GetString() << " " << mS2.GetString() << endl;
  mS1.Swap(mS2);
  cout << mS1.GetString() << " " << mS2.GetString() << endl;
}
```

```
class myString
{
private:
   char s[100];
public:
   char *GetString();
   void operator = (char const *);
};

void myString::operator = (char const *sNew)
{
    strcpy(s,sNew);
}

int main(){
  myString mS1, mS2;
  mS1 = "this";
  mS2 = "that";
  cout << mS1.GetString() << " " << mS2.GetString() << endl;
}
```

```
#include <iostream>

using namespace std;

int const SIZE = 100;

class Array{
private:
  int _array[SIZE];
public:
  int &operator [] (int i)
  {
    if (i < 0|| i >= SIZE){
      cout << "illegal access\n";
      exit(1);
    }
    return _array[i];
  }
};
```

Τελεστής που μπορεί να χρησιμοποιηθεί στα αριστερά μιας ανάθεσης.

```
int main(){
  Array A;
  A[10] = 1000;
  int x = A[10];
  cout << x;
}
```

Default τιμές στο πέρασμα παραμέτρων

```
void f (int x = 1);
void g (int a, int b = 0){
  cout << "a: " << a << " b: " << b << "\n";
}

int main ()
{
   f(5);
   f();
   g(1, 2);
   g(5);
}
```

Τι output θα έχουμε?

```
void f (int x){ …    //could be (int x =1 )
{
  cout << x << endl;
}
```

Default τιμές στο πέρασμα παραμέτρων

```
void f (int x = 1);
void g (int a = 0, int b){
  cout << "a: " << a << " b: " << b << "\n";
}

int main ()
{
   f(5);
   f();
   g(1, 2);
   g(5);
}
```

Τι output θα έχουμε?

```
void f (int x){ …    //could be (int x =1 )
{
  cout << x << endl;
}
```

Υπερφόρτωση συναρτήσεων

1
```
int divide(int a, int b){
  return a/b;
}
```

2
```
int divide(float a, float b){
  return (int)(a/b);
}
```

3
```
float divide(float a, float b){
  return a/b;
}
```

4
```
float divide(int a, int b){
  return ((float)a)/b;
}
```

Ποιοι συνδυασμοί ορισμών είναι δεκτοί?

1	4
2	3
1	2
1	3
2	4
3	4

```
#include <iostream>

using namespace std;

int const SIZE = 100;

class Array{
private:
  int _array[SIZE];
public:
  int &operator [] (int i)
  {
    if (i < 0|| i >= SIZE){
      cout << "illegal access\n";
      exit(1);
    }
    return _array[i];
  }
};
```

Κάνουμε την συνάρτηση του τελεστή να επιστρέφει μια αναφορά

```
int main(){
  Array A;
  A[10] = 1000;
  int x = A[10];
  cout << x;
}
```

Δομές (Structs)

- Ο πιο απλός τρόπος δήλωσης ενός struct είναι ως εξής:

```
struct structName {
  fieldType fieldName;
  fieldType fieldName;
  ...
};
```

- Στην C++ οι δομές μπορούν να έχουν και μεθόδους.

Κλάσεις και structs

- Κλάσεις και structs είναι σχεδόν πανομοιότυπες.
  - Υπάρχουν λεπτές διαφορές:
    - Π.χ., by default, οι μεταβλητές-μέλη μιας κλάσης είναι private, ενώ τα μέλη ενός struct είναι public.

- Προγραμματιστική πρακτική:
  - Structs χρησιμοποιούνται για την αποθήκευση μόνο δεδομένων (όχι συναρτήσεις), και όλες οι μεταβλητές είναι public.
  - Classes χρησιμοποιούνται για την αποθήκευση δεδομένων και ορισμό μεθόδων. Τα δεδομένα είναι private.

Αλφαριθμητικά

- Στην C++ τα strings είναι ξεχωριστοί τύποι δεδομένων.
- Ένα string είναι ένα αντικείμενο, και μπορούμε να καλέσουμε μεθόδους του αντικειμένου για να πάρουμε τα χαρακτηριστικά του string (π.χ., length) ή για να το επεξεργαστούμε.
  - Συγκριτικά, στην C, εφαρμόζαμε συναρτήσεις πάνω στα strings αντί να καλούμε μεθόδους του string.
  - Ο χειρισμός των strings γίνεται πολύ πιο εύκολος.

Strings – Δήλωση και αρχικοποίηση

```
#include <iostream>
#include <string>

using namespace std;
int main(){
 string imBlank;
 string heyMom(“where is my coat?”);
 string heyPap = “whats up?”;
 string heyGranPa(heyPap);
 string heyGranMa = heyMom;
}
```

Strings – Επεξεργασία

```
#include <iostream>
#include <string>
using namespace std;
int main(){
 string s1("I saw elvis in a UFO.");
 cout << s1.size() << \endl;
 cout << s1.length() << \endl;
 cout << s1.capacity() << \endl; // >= s1.length()
 string s2 = " thought I ";
 s1.insert(1, s2); // insert in position 1, i.e. right after 'I'
 cout << s1.capacity() << \endl;
 string s3 = "I've been working too hard";
 s1.append(s3);
 s1.append(", or am I crazy?");
 cout << s1 << endl;
 s1.resize(10); // increase/reduce the capacity
 cout << s1 << \endl;
}
```

Strings – Επεξεργασία με τελεστές

```
s= s1 + s2 + s3;
s += s5 + s6;
s[10] = 'c';
s.at(10) = 'c';
s1 == s2
s1 != s2
s1 >= s2
s1 <= s2
s1 > s2
s1 < s2
s1.compare(0,2,s2,0,2); // compare s1[0..2]
 with s2[0..2]
s1.swap(s2);
```

alphabetic ordering

Strings – Επεξεργασία

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

int main(){
  string s("Hello mother. How are you mother? Im fine mother.");
  string s1("bro");
  string s3("mo");
  int start = 0;

  int found = s.find(s3, start); // find the first occurrence of
 string s3 in string s, starting from start.

  while (found != string::npos){ // if s3 not in s, the function
 returns string::npos (the maximum possible string length)
    s.replace(found, s3.length(), s1); // makes "mo" to "bro"
    start = found + s1.length();
    cout << s << endl;
    found = s.find(s3, start);
  }
  cout << s << endl;
}
```

ΣΥΝΑΡΤΗΣΕΙΣ ΔΟΜΗΣΗΣ (ΚΑΤΑΣΚΕΥΗΣ) CONSTRUCTORS

Class Car

```
class Car
{

private:
   int _pos;

public:
   void InitializePosition();
   void Move();
   int GetPosition();
   bool Collide(Car)
};
```

Methods

```
void Car::InitializePosition()
{
   _pos = 0;
}
```

```
void Car::Move()
{
   _pos += floor((double(rand())/RAND_MAX)*3)-1;
}

int Car::GetPosition()
{
   return _pos;
}

bool Car::Collide(Car other)
{
    return (_pos == other.GetPosition());
}
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;
  carX.InitializePosition();
  carY.InitializePosition();

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Τι γίνεται αν ο προγραμματιστής ξεχάσει να κάνει την αρχικοποίηση?

Η μεταβλητή _pos θα πάρει τυχαία τιμή.

Constructors

- Μια μέθοδος που ο μόνος ρόλος της ειναι να κατασκευάζει (construct) και να αρχικοποιεί το αντικείμενο.
- Συντακτικό:
  - `<classname>()`
  - Μπορεί να έχει ορίσματα.
  - ΔΕΝ εχει τυπο επιστροφής – ΟΥΤΕ void.
- Υλοποίηση:
  - `<classname>::<classname>()`

- Η συνάρτηση καλείται αυτόματα με την δημιουργία του αντικειμένου.
  - Είτε στη δήλωση του αντικειμένου.
  - Είτε δημιουργία με new (θα το δούμε αργότερα).

Παράδειγμα

```
class Car
{

private:
   int _pos;

public:
   Car();
   void Move();
   int GetPosition();
   bool Collide(Car)
};
```

Παράδειγμα

```
Car::Car()
{
   _pos = 0;
}
```

Πιο σωστά:

```
Car::Car():_pos(0)
{
}
```

Αν είχαμε πολλές θέσεις να αρχικοποίησουμε:

```
Car::Car():_pos1(0),_pos2(9), _pos3(4), …
{
}
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Η αρχικοποίηση της θέσης γίνεται όταν ορίζουμε το αντικείμενο.

Υπερφόρτωση Constructor

- Τι γινεται αν θελουμε να δώσουμε την αρχική θέση από την είσοδο?
  - Υπερφόρτωση Constructor

```
class Car
{

private:
   int _pos;

public:
   Car();
   Car(int);
   void Move();
   int GetPosition();
   bool Collide(Car)
};
```

```
Car::Car():_pos(0)
{
}
```

```
Car::Car(int p):_pos(p)
{
}
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  int position;
  cin >> position;
  Car carY(position);

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Default Constructors

- Ανεξάρτητα του τι Constructors έχουμε ορίσει, μπορούμε πάντα να αρχικοποιήσουμε ένα αντικείμενο με ένα άλλο αντικείμενο.
  - Χρησιμοποιούμε τον Default Copy Constructor
  - Αρχικοποιεί όλα τα πεδία του αντικειμένου με τα πεδία του ορίσματος.
  - Προσοχή αν κάποια πεδία είναι δείκτες!

- Υπάρχει επίσης και ο Default Constructor ο οποίος δεν έχει ορίσματα και αρχικοποιεί τα πάντα σε 0 ή null.
  - Υπάρχει εφόσον δεν έχουμε ορίσει κάποιον άλλο constructor. Τον χρησιμοποιούσαμε εμμέσως σε όλα τα προηγούμενα παραδείγματα.
  - Αν ορίσουμε ένα constructor παύει να υπάρχει.

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  int position;
  cin >> position;
  Car carX(position);
  Car carY(carX);

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Constructors

- Οι constructors είναι πολύ πιο σημαντικοί (και βολικοί) όταν δεσμεύουμε μνήμη μέσα στον constructor.
  - Εξασφαλίζουμε ότι η μνήμη που χρειαζόμαστε για ένα αντικείμενο θα υπάρχει όταν το χρησιμοποιούμε.
  - Δεν εξαρτόμαστε από το να θυμηθεί ο προγραμματιστής να καλέσει την μέθοδο αρχικοποίησης.

myString

```
class myString
{
private:
   char s[100];
public:
   char *GetString();
   void SetString(char const *);
   void Swap(myString &);
};
```

Τι γίνεται αν δεν θέλουμε τα strings να είναι fixed size?

Το πλεονέκτημα: μπορούμε σε μια άλλη συνάρτηση να αλλάξουμε το capacity του string.

myString

```
class myString
{
private:
   char *s;
public:
   char *GetString();
   void SetString(char const *);
   void Swap(myString &);
};
```

Κανουμε τον πίνακα s να έχει δδυναμικά δεσμευόμενη μνήμη.

Το πλεονέκτημα: μπορούμε να ρυθμίσουμε δυναμικά το μέγεθος του string.

Το μειονέκτημα: πρέπει να φροντίζουμε για την δέσμευση μνήμης

myString

```
class myString
{
private:
   char *s;
public:
   myString();
   char *GetString();
   void SetString(char const *);
   void Swap(myString &);
};
```

```
myString::myString()
{
   s = new char[100];
};
```

Η δέσμευση της μνήμης θα γίνει μέσα στον constructor.

Η συνάρτηση δόμησης εξασφαλίζει ότι δεν θα δημιουργηθεί αντικείμενο που δεν έχει την απαραίτητη μνήμη.

Destructor

- Ότι δεσμεύεται θα πρέπει να αποδεσμεύεται.
- Ο Destructor αναλαμβάνει να καταστρέψει το αντικείμενο.
  - Αν δεν έχουμε δέσμευση μνήμης δεν είναι απαραίτητο να τον ορίσουμε, οι μεταβλητές μέλη καταστρέφονται όταν το αντικείμενο πάψει να υπάρχει.
  - Αν έχουμε δεσμεύσει μνήμη για το αντικείμενο θα πρέπει να την αποδεσμεύσουμε.

Destructor

- Μια μέθοδος που ο μόνος ρόλος της είναι να αποδομεί (destruct) το αντικείμενο.
  - Ο destructor κάνει ένα clean-up.
- Συντακτικό:
  - `~<classname>()`
  - ΔΕΝ μπορεί να έχει ορίσματα.
  - ΔΕΝ εχει τυπο επιστροφής – ΟΥΤΕ void.
- Υλοποίηση:
  - `<classname>::~<classname>()`

- Η συνάρτηση καλείται αυτόματα με την αποδόμηση του αντικειμένου.
  - Είτε γιατί παύει να υπάρχει (βγαίνουμε από το scope που είναι ορισμένο).
  - Είτε αποδόμηση με delete (θα το δούμε αργότερα).

myString

```
class myString
{
private:
   char *s;
public:
   myString();
   ~myString();
   char *GetString();
   void SetString(char const *);
   void Swap(myString &);
};


myString::myString()
{
   s = new char[100];
};
```

```
myString::~myString()
{
   delete [] s;
};
```

Η αποδέσμευση της μνήμης θα γίνει μέσα στον destructor.

Η αποδέσμευση της μνήμης είναι απαραίτητη για να αποφύγουμε memory leaks.

Default Destructor

- Τον χρησιμοποιούσαμε εμμέσως μέχρι τώρα. Δεν κάνει τίποτα. Δεν αποδεσμεύει μνήμη.

Προσοχή στα μέλη που είναι δείκτες!

- Με τον Default Copy Constructor, τα πεδία ενός αντικειμένου αντιγράφονται στο άλλο. Αν το πεδίο είναι ένας pointer τότε οι δύο pointers θα δείχνουν στην ίδια θέση μνήμης. Αυτό μπορεί να δημιουργήσει πρόβλημα:
  - Αν το ένα αντικείμενο καταστραφεί, ο destructor αποδεσμεύει τη μνήμη και ο pointer στο άλλο αντικείμενο δείχνει στο κενό (dangling pointer). Αυτό θα το δούμε όταν δημιουργούμε pointers σε αντικείμενα.
  - Παρόμοιο πρόβλημα μπορεί να δημιουργηθεί αν κάνουμε τον δείκτη του αντικειμένου να δείχνει σε μια θέση που μετά καταστρέφεται.

Προσοχή στα μέλη που είναι δείκτες!

```
myString::myString (char *y){
  s = y; // NOT A DEEP COPY
}
myString::~myString (){
   cout << s << " is deleted\n";
   delete[] s;
}
```

Υπερφόρτωση του Constructor με όρισμα string

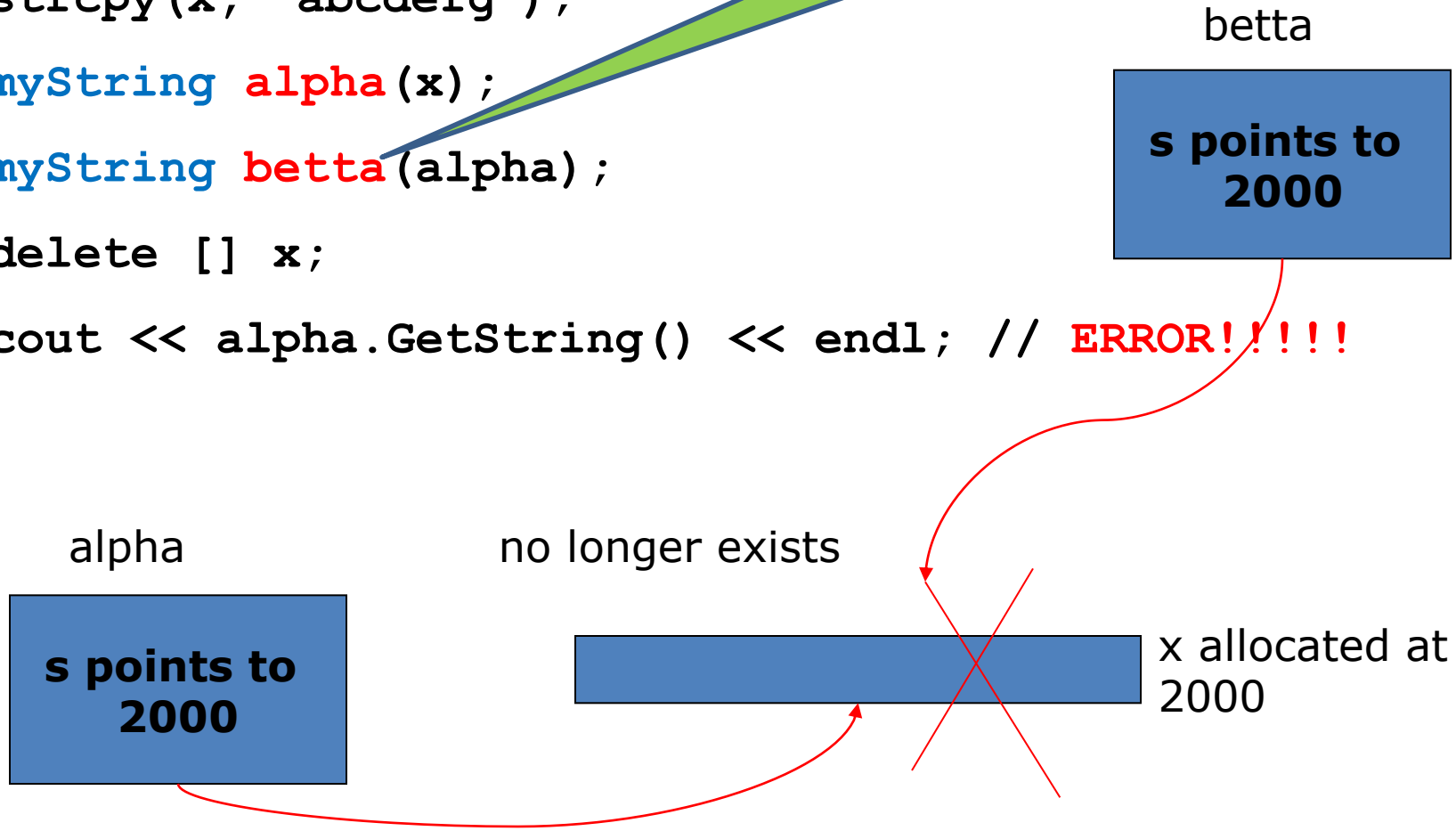
Υπάρχουν δυο πιθανά προβλήματα με αυτό τον κώδικα:

1. Ο χώρος μνήμης στον οποίο δείχνει το **y** αποδεσμεύεται και το **s** γίνεται dangling pointer
2. Το αντικείμενο καταστρέφεται και ο χώρος αποδεσμεύεται και η παράμετρος **y** γίνεται dangling pointer

Παράδειγμα με το πρόβλημα 2.

```
main(){
  char *x = new char[10];
  strcpy(x, "abcdefg");
  myString alpha(x);
  myString betta(alpha);
  delete [] x;
  cout << alpha.GetString() << endl; // ERROR!!!!!
}
```

Κατασκευή του betta με default copy constructor



Μία λύση

```
myString::myString(char *y){
    cout << y << " is copied\n";
    s = new char [strlen(y)+1];
    strcpy(s,n);  // DEEP COPY HERE
}

myString::myString(myString &x){
  s = new char [strlen(x.GetString())+1];
  strcpy(s, x.GetString()); // DEEP COPY
}

myString::~myString(){
    cout << s << " is deleted\n";
    delete[] s;
}
```

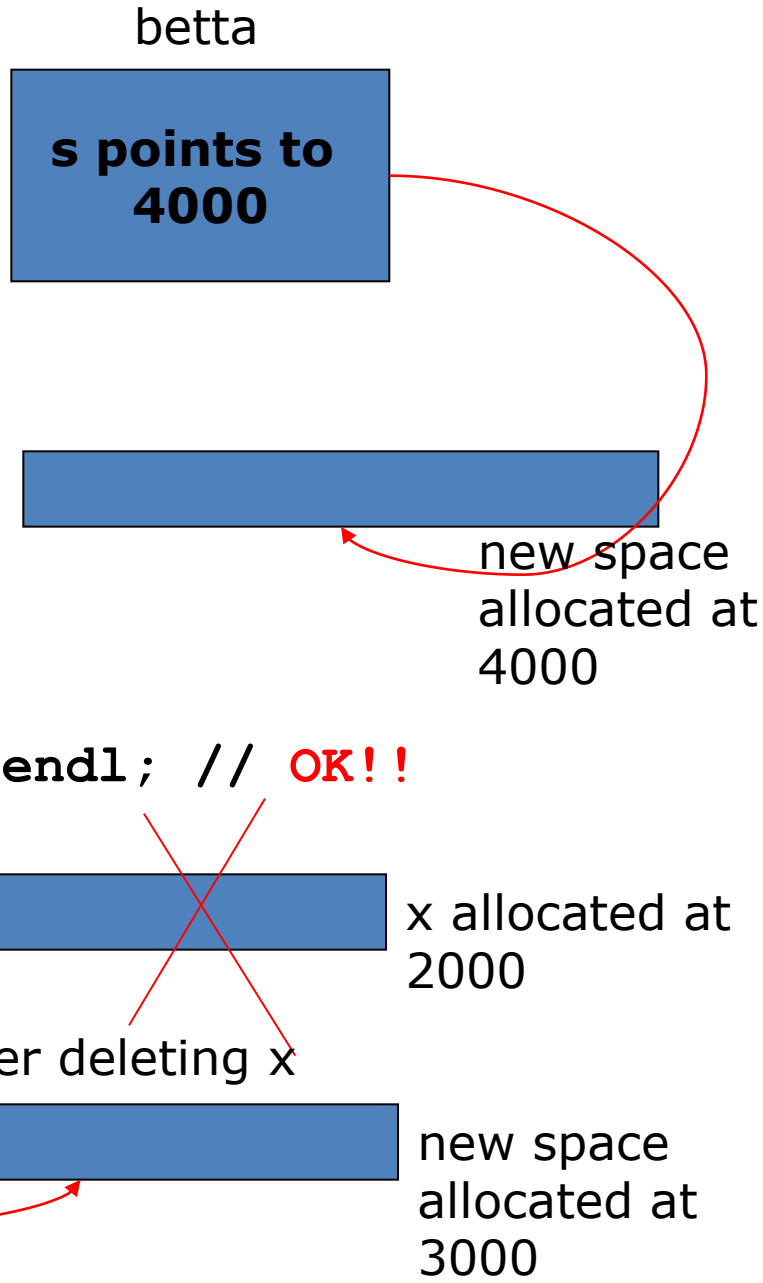
Δημιουργία τοπικά χώρου και αντιγράφου του string. Το **s** δεν σχετίζεται πια με το χωρο μνημης του **y**.

Υπερφόρτωση του Default Copy Constructor

Ο destructor ελευθερώνει τον τοπικό χώρο μνήμης και η παράμετρος **y** δεν επηρεάζεται

Παραδειγμα

```
main(){
 char *x = new char[10];
 strcpy(x, “abcdefg”);
 myString alpha(x);
 myString betta(alpha);
 delete [] x;
 cout << alpha.GetString() << endl; // OK!!
}
```



Παράδειγμα

```
class intArray
{
private:
    int *A;
public:
    intArray();
    ~intArray();
    void cleanArray();
};

intArray::intArray()
{
    A = new int[10];
}

intArray::~intArray()
{
    delete [] A;
}
```

```
void intArray::cleanArray()
{
    delete [] A;
}

int main()
{
    intArray X;
    X.cleanArray();
}
```

Το πρόγραμμα αυτό προκάλει memory error, γιατι?

Η κλήση της μεθόδου `x.cleanArray()` αποδεσμεύει τη μνήμη του πίνακα `A` μέσα στο αντικείμενο `X`.
Όταν το πρόγραμμα ολοκληρωθεί το αντικείμενο `X` καταστρέφεται και καλείται αυτόματα ο destructor ο οποίος προσπαθεί να σβήσει ξανά το `A` προκαλώντας λάθος.