Δείκτες

Εργίνα Καβαλλιεράτου

Σημερινό Μάθημα

- ✓Πίνακες
- ✓Strings
- ✓Δείκτες
- ✓Τελεστές *, &
- ✓Ανάκληση
- ✓Δέσμευση μνήμης new / delete
- ✓Επαναδέσμευση
- ✓Δείκτες ως παράμετροι

Πίνακες

```
#include <iostream.h>
 int main(){
 int myArray[5];
 int i;
 for ( i=0; i<5; i++)  {
      cout << "Value for myArray[" << i << "]: ";
      cin >> myArray[i]; }
 for (i = 0; i<5; i++)
      cout << i << ": " << myArray[i] << "\n";
 return 0;}
```

```
Value for myArray[0]: 3
Value for myArray[1]: 6
Value for myArray[2]: 9
Value for myArray[3]: 12
Value for myArray[4]: 15
0: 3
1: 6
2: 9
3: 12
4: 15
```

Πολυδιάστατοι Πίνακες

```
#include <iostream.h>
  int main(){
    int SomeArray[5][2] = { {0,0}, {1,2}, {2,4}, {3,6}, {4,8}};
    for (int i = 0; i<5; i++)
          for (int j=0; j<2; j++){
             cout << "SomeArray[" << i << "][" << j << "]: ";
             cout << SomeArray[i][j]<< endl; }
    return 0;
  }
```

```
SomeArray[0][0]: 0
SomeArray[0][1]: 0
SomeArray[1][0]: 1
SomeArray[1][1]: 2
SomeArray[2][0]: 2
SomeArray[2][1]: 4
SomeArray[3][0]: 3
SomeArray[3][1]: 6
```

String

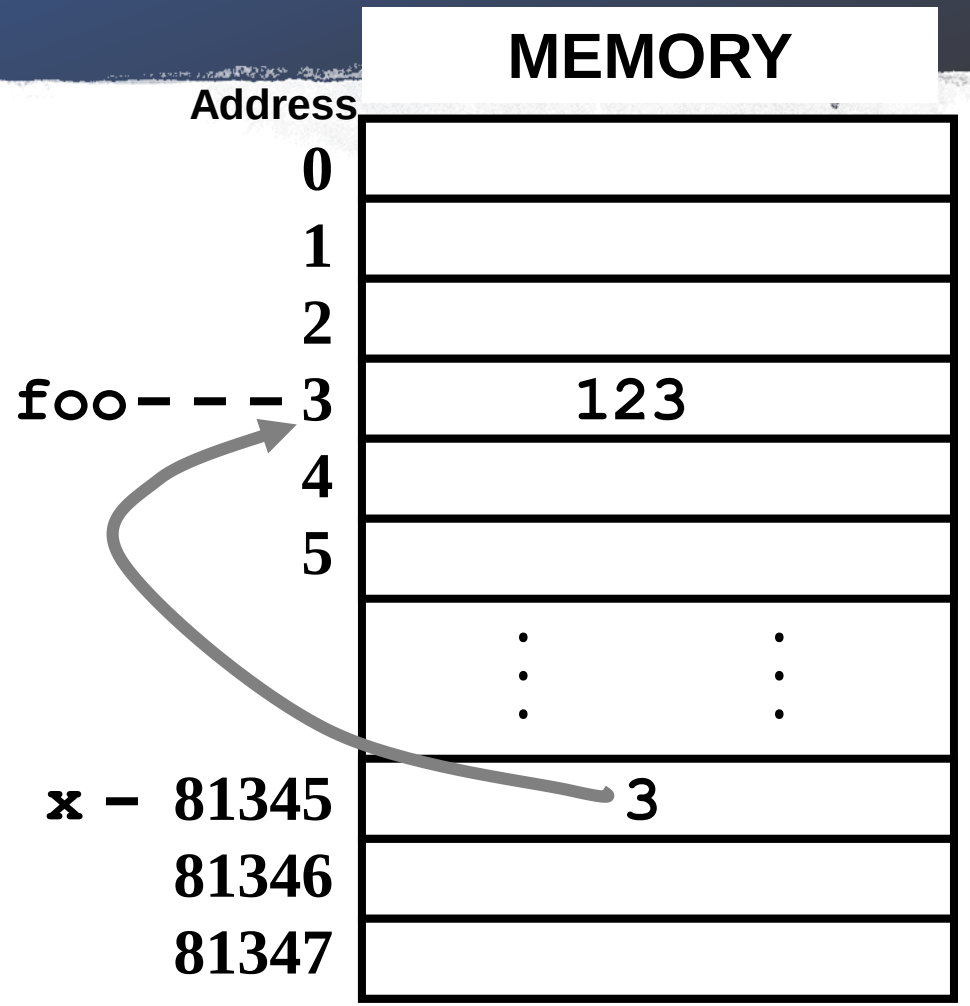
```
#include <iostream.h>
 int main()
 {
   char buffer[80];
   cout << "Enter the string: ";
   cin >> buffer;
   cout << "Here's the buffer: " << buffer << endl;
   return 0;
 }
```

```
Enter the string: Hello World
Here's the buffer: Hello
```

Χρήση pointer

```
int foo;
int *x;

foo = 123;
x = &foo;
```



Χρήση pointer

```
int *pAge = 0;
```

- Ο pAge είναι δείκτης σε ακέραιο δηλαδή θα έχει τη διεύθυνση μιας θέσης μνήμης αρκετά μεγάλης για να αποθηκεύσει ακέραιο.
- Όταν η διεύθυνση ενός δείκτη είναι 0 κατά σύμβαση ο δείκτης είναι null.
- Καλό είναι οι δείκτες να αρχικοποιούνται κατά τη δήλωση τους:

```
unsigned short int howOld = 50;
unsigned short int * pAge = &howOld;
```

- Ο δείκτης παρέχει έμμεση πρόσβαση στη μεταβλητή, της οποίας τη διεύθυνση περιέχει:

```
unsigned short int yourAge;
yourAge = *pAge;
```

Τελεστής *

- Ο τελεστής * έχει δύο χρήσεις:

```
unsigned short * pAge = 0;

*pAge = 5;
```

Τελεστής &

- Για να πάρουμε τη διεύθυνση μιας μεταβλητής χρησιμοποιούμε τον τελεστή &

```
int a = 3, *ptr=0;
ptr = &a;
cout << "The address of variable a is " << (int)ptr << " and it contains "
  << *ptr << endl;
```

- Ο τύπος του δείκτη ορίζει το πλήθος των bytes που απαιτεί η μεταβλητή ΌΧΙ ο δείκτης.

Παράδειγμα

```
#include <iostream.h>:
  typedef unsigned short int USHORT;
  int main()  {
     USHORT myAge;
     USHORT * pAge = 0;
     myAge = 5;
     cout << "myAge: " << myAge << "\n";
     pAge = &myAge;
     cout << "*pAge: " << *pAge << "\n\n";
      cout << "*pAge = 7\n";
*pAge = 7;
 cout << "*pAge: " << *pAge << "\n";
 cout << "myAge: " << myAge << "\n\n";
 cout << "myAge = 9\n";
 myAge = 9;
 cout << "myAge: " << myAge << "\n";
 cout << "*pAge: " << *pAge << "\n";
 return 0;
}
```

```
myAge: 5
*pAge: 5

*pAge = 7
*pAge: 7
myAge: 7

myAge = 9
myAge: 9
*pAge: 9
```

Γενικοί δείκτες: void*

- Δήλωση: **void * ptr;**
- Μπορεί να δείξει σε οποιοδήποτε τύπο

```
void * ptr = GetAddress();
float *fptr = (float*)ptr;
```

- Οι συναρτήσεις μπορούν να λάβουν και να επιστρέψουν δείκτες void
- ΟΛΟΙ οι δείκτες έχουν το ίδιο μέγεθος

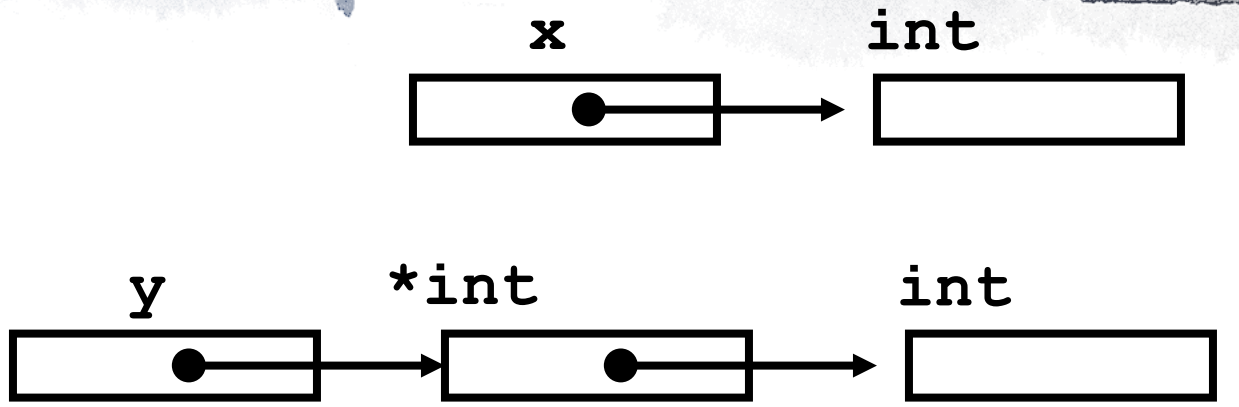
Ανάκληση

- Μπορεί να χρησιμοποιηθεί δείκτης που δείχνει σε δείκτη
- Λέγεται Ανάκληση

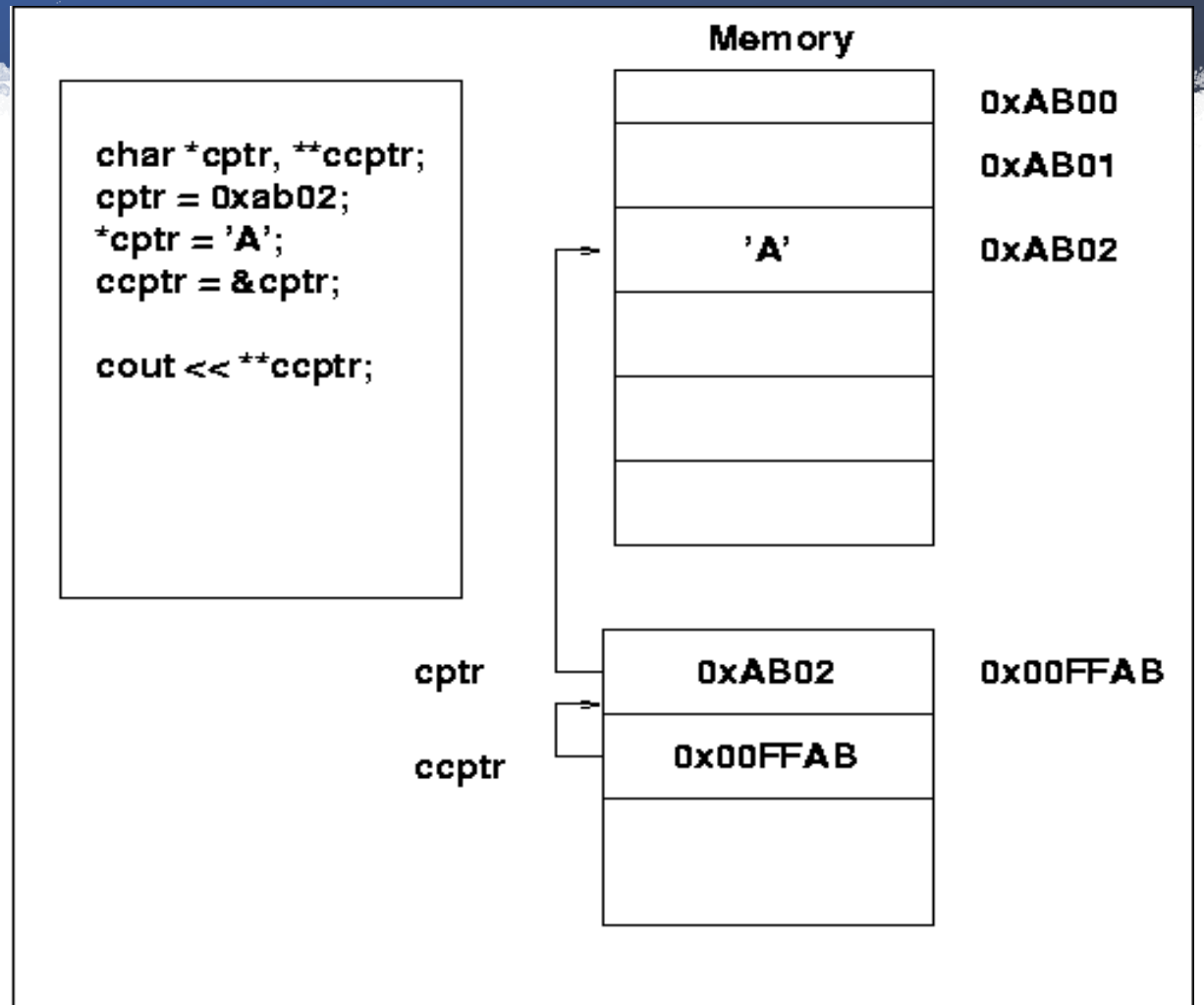
  char **ccptr;

Ανάκληση

```
int *x;
int **y;
```



Ανάκληση


Memory
char *cptr, **ccptr;
cptr = 0xab02;
*cptr = 'A';
ccptr = &cptr;
cout << **ccptr;
0xAB00
0xAB01
'A'
0xAB02
cptr
ccptr
0xAB02
0x00FFAB
0x00FFAB

Χρησιμότητα pointer

- ✓Διαχείριση δεδομένων & μνήμης (δέσμευση μνήμης)
- ✓Πρόσβαση στα δεδομένα-μέλη κλάσεων και συναρτήσεων
- ✓Πέρασμα με αναφορά σε συναρτήσεις

Δέσμευση μνήμης/new

- Για να διαχειριστούμε σωστά τα δεδομένα και να δεσμεύσουμε μνήμη όπου χρειάζεται χρησιμοποιούμε την εντολή new:

```
unsigned short int * pPointer;

pPointer = new unsigned short int;
```

Ή

```
unsigned short int * pPointer = new unsigned short int;
```

- Το new ακολουθείται από τον τύπο του αντικειμένου που θέλουμε να δεσμεύσουμε
- Η διεύθυνση αποδίδεται σε ένα δείκτη
- Κάθε φορά που αιτούμαστε για μνήμη πρέπει να ελέγχουμε το δείκτη για null (αποτυχία στη δέσμευση)

delete

- Όταν δεν μας χρειάζεται η δεσμευμένη μνήμη την απελευθερώνουμε με delete:

```
delete pPointer;
```

- Αν ξανακαλέσουμε το delete στον ίδιο δείκτη μπορεί να προκαλέσει crash
- Για τον ίδιο λόγο καλό είναι μετά το delete να αποδίδουμε στο δείκτη τιμή 0(null)
- Crash θα προκαλέσει και η προσπάθεια να αποδώσουμε εκ νέου μνήμη σε διαγραμμένο δείκτη.
- ΠΡΟΣΟΧΗ: το σωστό είναι για κάθε new να υπάρχει ένα delete

Επαναδέσμευση

Λάθος

```
unsigned short int * pPointer = new unsigned short int;
*pPointer = 72;
pPointer = new unsigned short int;
*pPointer = 84;
```

Σωστό

```
unsigned short int * pPointer = new unsigned short int;
*pPointer = 72;
delete pPointer;
pPointer = new unsigned short int;
*pPointer = 84;
```

Παράδειγμα/1

```
#include <iostream.h>
  int main()
  {
    int localVariable = 5;
    int * pLocal= &localVariable;
    int * pHeap = new int;
    if (pHeap == NULL)       {
        cout << "Error! No memory for pHeap!!";
        return 0;       }
    *pHeap = 7;
    cout << "localVariable: " << localVariable << "\n";
    cout << "*pLocal: " << *pLocal << "\n";
```

```
localVariable: 5
*pLocal: 5
```

Παράδειγμα/2

```
    cout << "*pHeap: " << *pHeap << "\n";
    delete pHeap;
    pHeap = new int;
    if (pHeap == NULL)      {
        cout << "Error! No memory for pHeap!!";
        return 0;     }
    *pHeap = 9;
    cout << "*pHeap: " << *pHeap << "\n";
    delete pHeap;
    return 0;
}
```

```
localVariable: 5
*pLocal: 5
*pHeap: 7
*pHeap: 9
```

Pointer σε αντικείμενο

Cat *pCat = new Cat;

Γίνεται δέσμευση για όσο χώρο απαιτεί η συγκεκριμένη κλάση

Παράδειγμα /1

```
#include <iostream.h>
  class SimpleCat
  {
  public:
        SimpleCat();
         ~SimpleCat();
   private:
         int itsAge;
    };
```

```
SimpleCat::SimpleCat()
      {
            cout << "Constructor called.\n";
            itsAge = 1;
      }
```

Παράδειγμα /2

```
SimpleCat::~SimpleCat()
    {
        cout << "Destructor called.\n";
    }
```

```
SimpleCat Frisky...
Constructor called.
SimpleCat *pRags = new SimpleCat..
Constructor called.
delete pRags...
Destructor called.
Exiting, watch Frisky go...
Destructor called.
```

```
int main()
{
    cout << "SimpleCat Frisky...\n";
    SimpleCat Frisky;
    cout << "SimpleCat *pRags = new SimpleCat...\n";
    SimpleCat * pRags = new SimpleCat;
    cout << "delete pRags...\n";
    delete pRags;
    cout << "Exiting, watch Frisky go...\n";
    return 0;
}
```

Παράδειγμα 1.1

```
#include <iostream>
using namespace std;
int main (){
  int firstvalue, secondvalue;
  int * mypointer;
  mypointer = &firstvalue;
  *mypointer = 10;
  mypointer = &secondvalue;
  *mypointer = 20;
  cout << "firstvalue is " << firstvalue << '\n';
  cout << "secondvalue is " << secondvalue << '\n';
  return 0;}
```

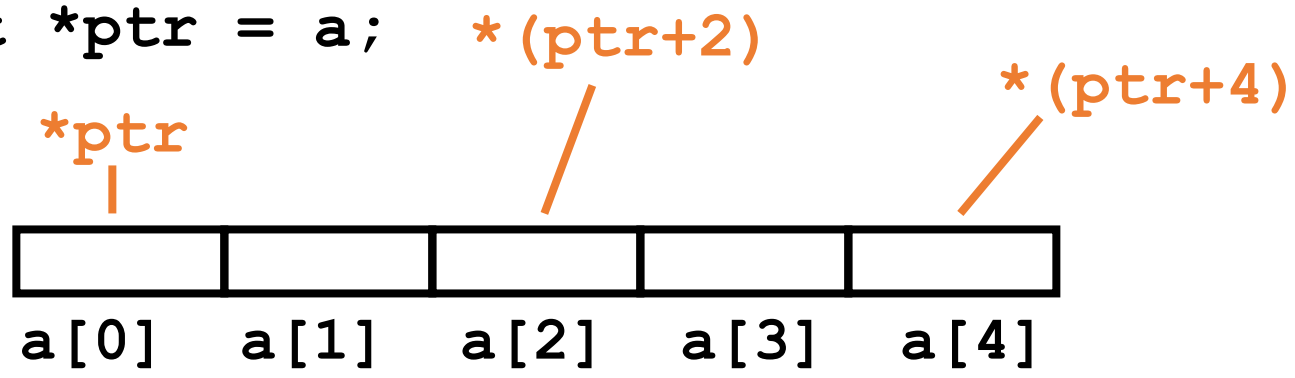
Παράδειγμα 2.1

```
#include <iostream>
int main (){
  int firstvalue = 5, secondvalue = 15;
  int * p1, * p2;
  p1 = &firstvalue;    p2 = &secondvalue;
  *p1 = 10;   *p2 = *p1;
  p1 = p2;   *p1 = 20;
  cout << "firstvalue is " << firstvalue << '\n';
  cout << "secondvalue is " << secondvalue << '\n';
  return 0; }
```

Δείκτες (Pointers)

- Οι δείκτες μπορούν να χρησιμοποιηθούν σε μαθηματικές εκφράσεις
- Όταν αυξάνουμε ένα δείκτη θα αυξηθεί σύμφωνα με το μέγεθος του αντικειμένου που δείχνει

```
int *ptr = a;
```




```
int a[5];
```

Δείκτες και πίνακες

- Το όνομα ενός πίνακα είναι στη ουσία ένας *const* δείκτης.

```
int *x=0;
int a[10];
x = &a[2];
for (int  i=0;i<3;i++)
   x[i]++;
```

Το **x** δείχνει στο **a[2]**

Το **x[i]** είναι το ίδιο με **a[2+i]**

Παράδειγμα 3.1

```
#include <iostream>
int main (){
  int numbers[5];  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;}
```

Εκτύπωση πίνακα

```
void print_array(int a[], int len) {
 for (int i=0;i<len;i++)
     cout << "[" << i << "] = "
          << a[i] << endl;
}
```

array version

```
void print_array(int *a, int len) {
  for (int i=0;i<len;i++)
      cout << "[" << i << "] = "
           << *a++ << endl;
}
```

pointer version

Παράδειγμα

```
//=================================== swap_ref
void swap_ref(int& a, int& b)
{
	int temp = a;
	a = b;
	b = temp;
}
//=================================== swap_ptr
void swap_ptr(int* a, int* b)
{
	 int temp = *a;
	*a = *b;
	*b = temp;
}
```

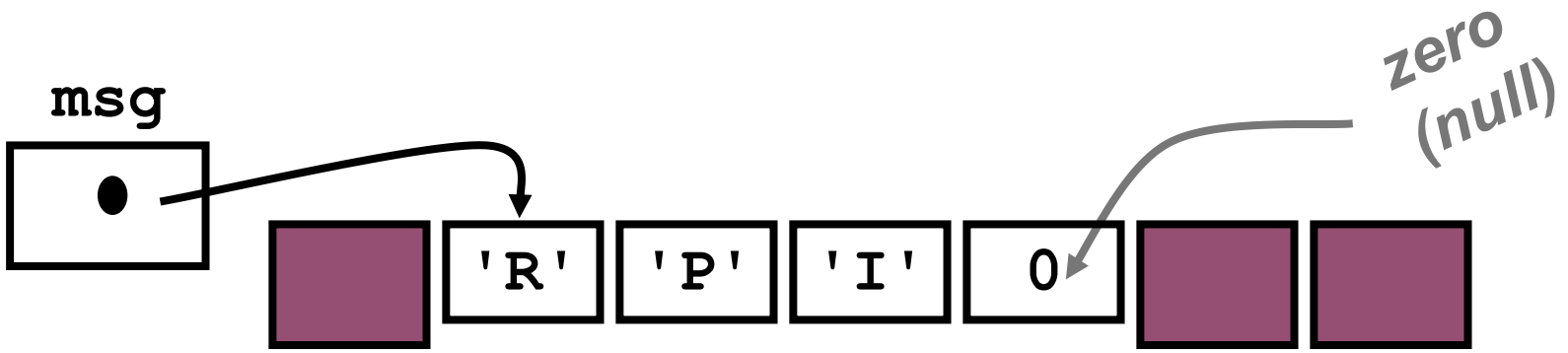
Δείκτες ως παράμετροι

- Οι δείκτες περνούν με τιμή(?)
- Αν αλλάξουμε αυτό που δείχνει θα επιστραφεί η νέα τιμή

Δείκτες και strings

- Οι δείκτες χρησιμοποιούνται συχνά με strings:

```
char *msg = "RPI";
```



strcpy

```
void strcpy(char a[], const char b[])
{
	int i = 0;
	while (b[i] != '\0') {
		a[i] = b[i]; i++;
	}
	 a[i] = '\0';
}
```

strcpy

```
void strcpy(char* a, char* b) {
	while (*b != '\0') {
		*a = *b;
		a++;
		b++;
	}
	*a = '\0';
}
```

'H

```
void strcpy(char* a, char* b) {while (*a++ = *b++);}
```

Pointer σε κλάση

- Όταν ορίσουμε pointer σε κλάση, θα πρέπει να χρησιμοποιείται μετά ως:

```
(*pRags).GetAge();
```

- Για ευκολία όμως η C++ επιτρέπει ισοδύναμα:

```
pRags->GetAge();
```

Δέσμευση μνήμης σε κλάση/1

```
#include <iostream.h>
  class SimpleCat    {
  public:
        SimpleCat();          ~SimpleCat();
        int GetAge() const { return *itsAge; }
        void SetAge(int age) { *itsAge = age; }
        int GetWeight() const { return *itsWeight; }
        void setWeight (int weight) { *itsWeight = weight; }
  private:
        int * itsAge;
        int * itsWeight;      };
```

Δέσμευση μνήμης σε κλάση/2

```
SimpleCat::SimpleCat()  {
    itsAge = new int;
    itsWeight = new int;  }
SimpleCat::~SimpleCat() {
    delete itsAge;       delete itsWeight;  }
int main()    {
     SimpleCat *Frisky = new SimpleCat;
     cout << "Frisky is " << Frisky->GetAge() << " years old\n";
     Frisky->SetAge(5);
     cout << "Frisky is " << Frisky->GetAge() << " years old\n";
     delete Frisky;       return 0;}
```

Δείκτης this/2

```
#include <iostream.h>
   class Rectangle
   {
    public:
        Rectangle();          ~Rectangle();
        void SetLength(int length) { this->itsLength = length; }
        int GetLength() const { return this->itsLength; }
        void SetWidth(int width) { itsWidth = width; }
        int GetWidth() const { return itsWidth; }
    private:
        int itsLength;          int itsWidth;      };
```

Δείκτης this/2

```
Rectangle::Rectangle()    {
    itsWidth = 5;          itsLength = 10;    }
Rectangle::~Rectangle()      {}
 int main()   {
     Rectangle theRect;
     cout << "theRect is " << theRect.GetLength() << " feet long.\n";
     cout << "theRect is " << theRect.GetWidth() << " feet wide.\n";
     theRect.SetLength(20);
     theRect.SetWidth(10);
     cout << "theRect is " << theRect.GetLength()<< " feet long.\n";
     cout << "theRect is " << theRect.GetWidth()<< " feet wide.\n";      return 0; }
```

???

```
#include <iostream.h>
int main()
{     int *pInt;
    *pInt = 9;
    cout << "The value at pInt: " << *pInt;
    return 0;
}
```

???

```
int main()
{
    int SomeVariable = 5;
    cout << "SomeVariable: " << SomeVariable << "\n";
    int *pVar = & SomeVariable;
    pVar = 9;
    cout << "SomeVariable: " << *pVar << "\n";
return 0;
}
```