

# Δείκτες

#9

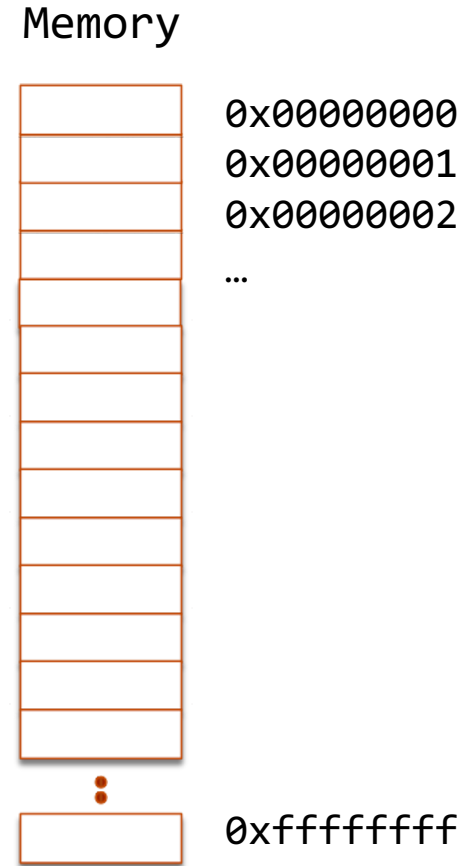
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Πανεπιστήμιο Ιωαννίνων (Άρτα)

Γκόγκος Χρήστος

# Μνήμη

- Η κύρια μνήμη του υπολογιστή χρησιμοποιείται για να αποθηκεύει κώδικα (το ίδιο το πρόγραμμα) και μεταβλητές του προγράμματος.
- Η μνήμη μπορεί να θεωρηθεί ως ένας τεράστιος πίνακας.
- Η αναφορά σε κάθε byte της μνήμης μπορεί να γίνει με τη χρήση μιας «διεύθυνσης» (σε έναν 32-bit υπολογιστή η διεύθυνση είναι 32 bits, σε έναν 64-bit υπολογιστή η διεύθυνση είναι 64 bits)



# Μνήμη

- Κάθε μεταβλητή αποθηκεύεται κάπου στη μνήμη, συσχετίζεται δε με μια διεύθυνση μνήμης που επιτρέπει την πρόσβαση στη μεταβλητή.
- Διάφοροι τύποι μεταβλητών καταλαμβάνουν διαφορετικό χώρο στη μνήμη:
  - char (1 byte)
  - int (4 bytes)
  - float (4 bytes)
  - double (8 bytes)
- Η συνάρτηση `sizeof()` επιστρέφει το μέγεθος κάθε μεταβλητής.
  - Δείτε το `sizeof.cpp`

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

int main() {
    int i;
    char c;
    char *ptr = &c;
    cout << "size of int variable = " << sizeof(i) << endl;
    cout << "size of char variable = " << sizeof(c) << endl;
    cout << "size of char* variable = " << sizeof(ptr) << endl;
    return 0;
}
```

```
size of int variable = 4
size of char variable = 1
size of char* variable = 8
```

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sizeof.cpp>

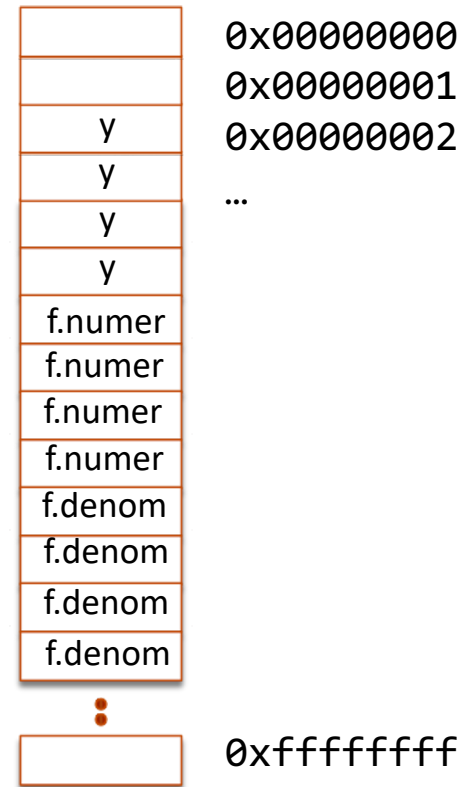
# Μνήμη

- Τα μέλη δεδομένων ενός αντικειμένου αποθηκεύονται και αυτά στη μνήμη.

```
main()  
{  
  int y;  
  Fraction f;  
  ...  
}
```

- Η μνήμη χωρίζεται σε διαφορετικά τμήματα:
  - Η κάθε μεταβλητή αποθηκεύεται καταλαμβάνοντας διαφορετική περιοχή.
  - στατική μνήμη** (static memory): καθολικές μεταβλητές
  - μνήμη στοίβας** (stack memory): τοπικές μεταβλητές
  - μνήμη σωρού** (heap memory): δυναμικά δεσμευμένη μνήμη

## Memory

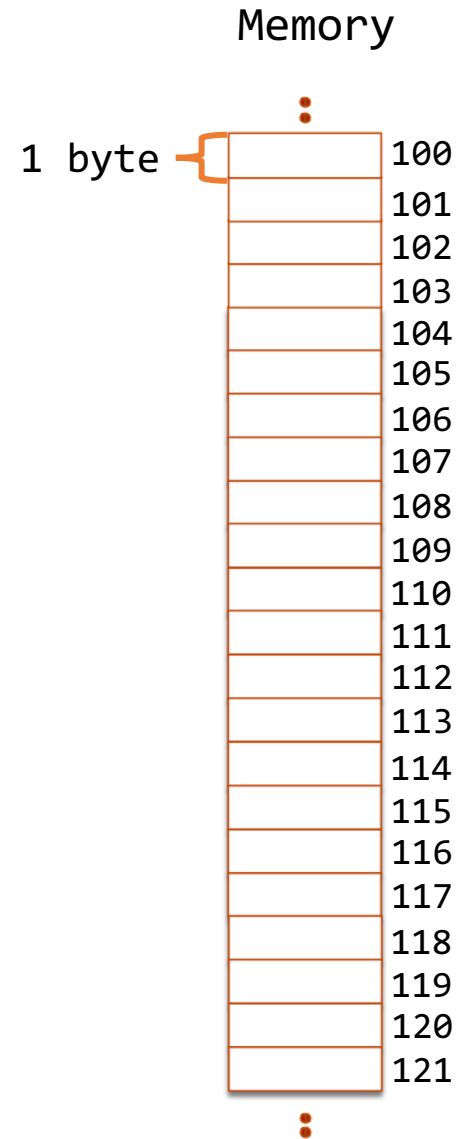


# Δείκτες

- Κάθε μεταβλητή (αντικείμενο, μέλος δεδομένων, κ.λπ.) αποθηκεύεται σε μια θέση στη μνήμη.
- Κάθε θέση μνήμης αντιστοιχείται σε έναν μοναδικό αριθμό που ονομάζεται διεύθυνση της θέσης μνήμης.
- Ένας δείκτης είναι μια μεταβλητή που περιέχει μια θέση μνήμης.
- Ένας δείκτης μπορεί να χρησιμοποιηθεί για να αποθηκεύσει τη θέση ενός αντικειμένου ή μιας μεταβλητής στη μνήμη.
- Στη συνέχεια μπορούμε να κάνουμε «αποαναφορά» (dereference) ενός δείκτη έτσι ώστε να αποκτήσουμε απευθείας πρόσβαση στο αντικείμενο ή στην μεταβλητή που δείχνει ο δείκτης.

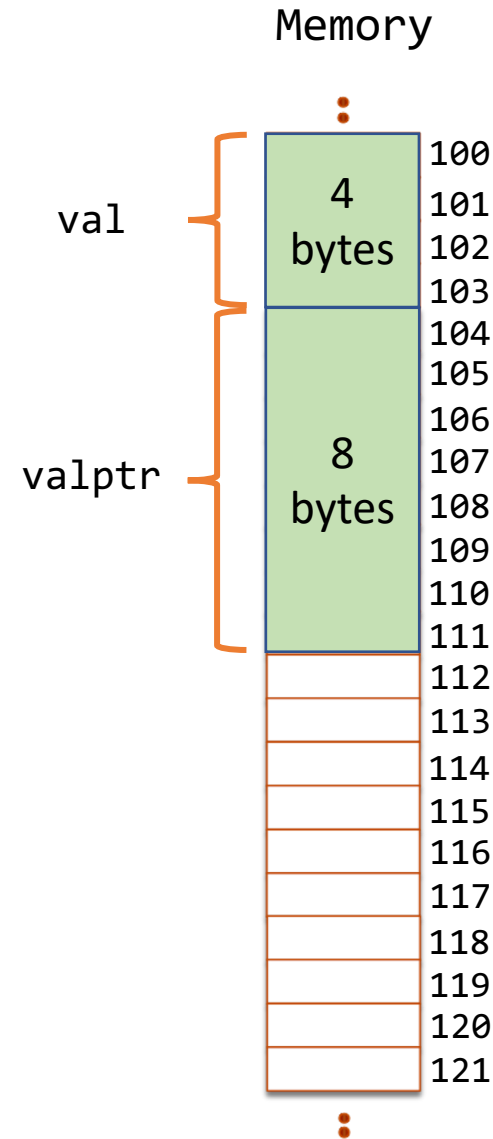
# Δείκτες

- Κάθε διεύθυνση μνήμης αναφέρεται στη θέση ενός απλού byte δεδομένων και οι συνεχόμενες διευθύνσεις μνήμης αναφέρονται σε συνεχόμενα bytes.
- Η μνήμη μπορεί να θεωρηθεί ως ένας τεράστιος πίνακας χαρακτήρων.



# Δείκτες

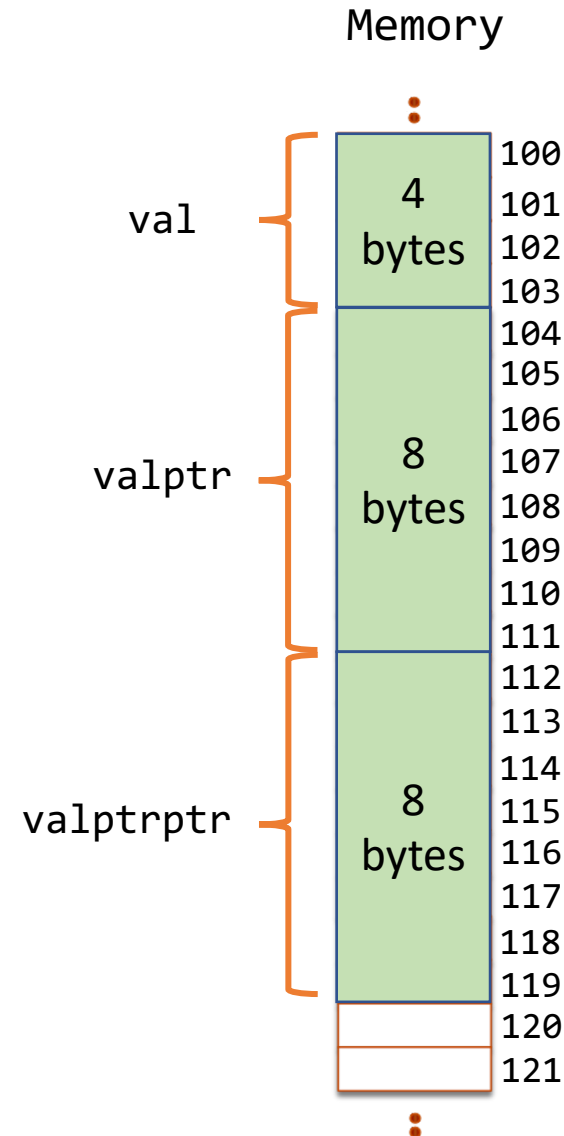
- Ένας δείκτης καταλαμβάνει χώρο στη μνήμη, όπως οποιαδήποτε άλλη μεταβλητή.
- Καθώς ένας δείκτης κρατά μια διεύθυνση και τα 64bit συστήματα χρησιμοποιούν διευθύνσεις 64bits, χρειάζονται 8bytes για να αναπαραστήσει κάθε διεύθυνση ( $64\text{bits} * 1\text{byte}/8\text{bits} = 8\text{ bytes}$ ).
- Όλοι οι δείκτες καταλαμβάνουν τον ίδιο χώρο μνήμης ανεξάρτητα από τον τύπο τους (`char*`, `int*`, `double*`)



```
void foo() {  
    int val;  
    int *valptr;  
    ...  
}
```

# Δείκτες

- Ένας δείκτης σε δείκτη κρατά τη διεύθυνση ενός δείκτη του ίδιου τύπου και συνεπώς καταλαμβάνει τον ίδιο χώρο μνήμης όπως ένας δείκτης.
- Ένας δείκτης σε ακέραιο, `int*`, κρατά τη διεύθυνση στην οποία βρίσκεται ένας `int`.
  - Ένας δείκτης `int**`, κρατά τη διεύθυνση στην οποία βρίσκεται ένας δείκτης `int*`.
    - Ένας δείκτης `int***`, κρατά τη διεύθυνση στην οποία βρίσκεται ένας δείκτης `int**`.
      - Κ.Ο.Κ.

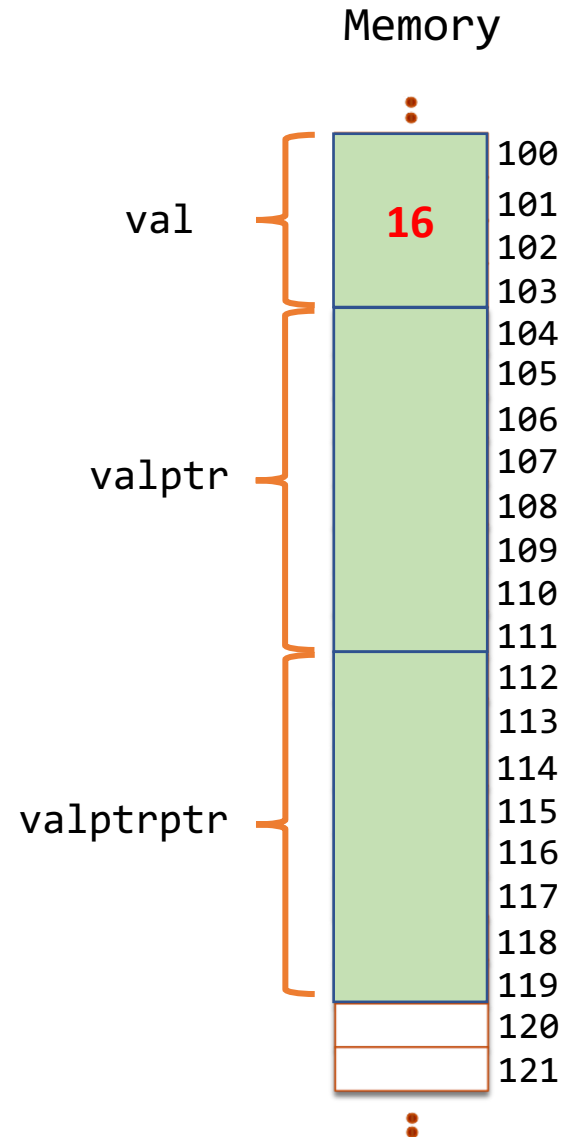


```
void foo() {  
    int val;  
    int *valptr;  
    int **valptrptr;  
    ...  
}
```



# Δείκτες

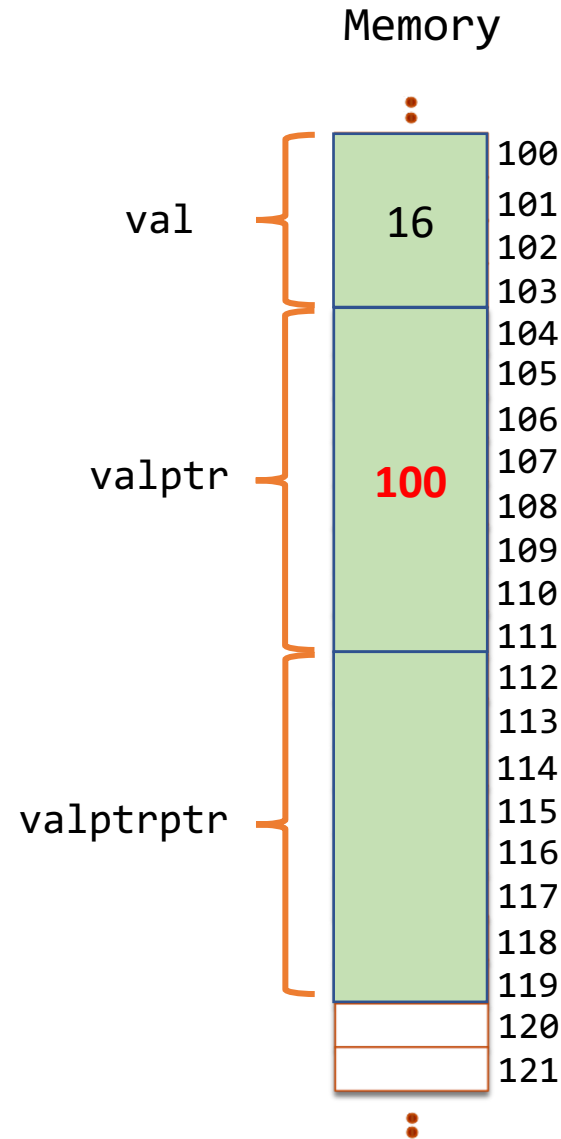
- Όταν ο μεταγλωττιστής πραγματοποιεί μια ανάθεση τιμής, μεταβαίνει στη διεύθυνση μιας μεταβλητής και ενημερώνει την τιμή που είναι αποθηκευμένη σε αυτή.
- Γενικότερα, όταν ο προγραμματιστής γράφει το όνομα *x* μιας μεταβλητής, αυτό μπορεί να ερμηνευτεί ως «η τιμή στη θέση μνήμης που έχει δεσμευτεί για το *x*».



```
void foo() {  
    int val;  
    int *valptr;  
    int **valptrptr;  
  
    val = 16;  
    valptr = &val;  
    *valptr = 5;  
  
    valptrptr = &valptr;  
    *valptrptr = NULL;  
}
```

# Δείκτες

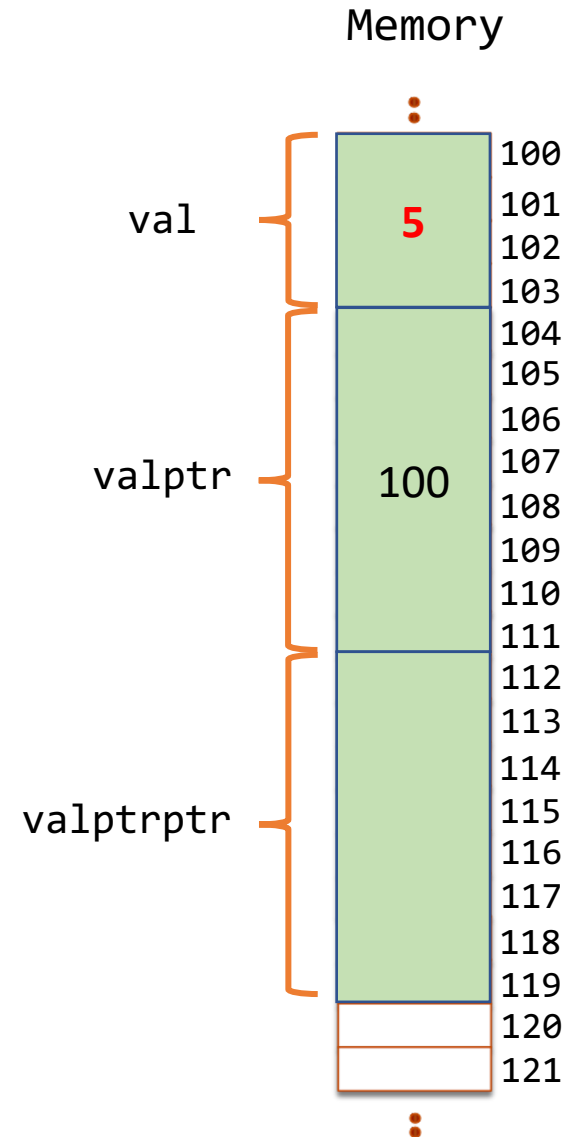
- Τοποθετώντας τον τελεστή & εμπρός από το όνομα μιας μεταβλητής, μπορούμε να αναφερθούμε στη διεύθυνση μιας μεταβλητής αντί για την τιμή της.
- Συνεπώς &x σημαίνει «η διεύθυνση της θέσης μνήμης που έχει δεσμευθεί για το x».
- Η διεύθυνση της μεταβλητής είναι η διεύθυνση του πρώτου byte που καταλαμβάνει στη μνήμη.



```
void foo() {  
    int val;  
    int *valptr;  
    int **valptrptr;  
  
    val = 16;  
    valptr = &val;  
    *valptr = 5;  
  
    valptrptr = &valptr;  
    *valptrptr = NULL;  
}
```

# Δείκτες

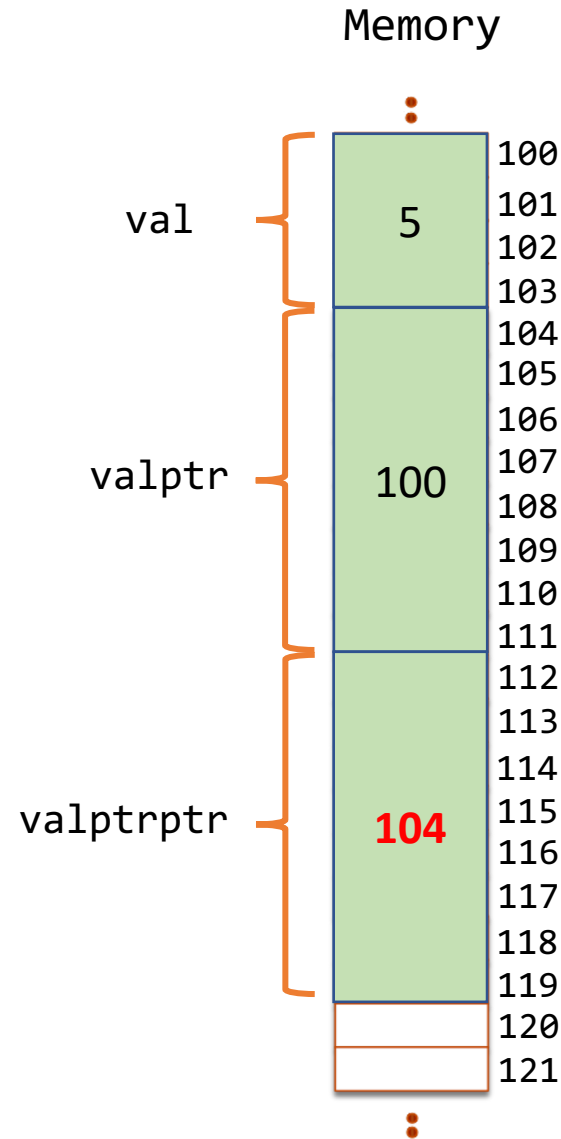
- Για μεταβλητές που είναι δείκτες, ο τελεστής \* επιτρέπει στον προγραμματιστή να προσπελάσει την τιμή της διεύθυνσης μνήμης που περιέχει ο δείκτης.
- Η διαδικασία αυτή καλείται αποαναφορά (dereferencing) του δείκτη και είναι επίσης γνωστή ως έμμεση αναφορά (indirection).
- Η έμμεση αναφορά είναι ο λόγος για τον οποίο οι δείκτες έχουν τύπο, καθώς αν και όλοι οι δείκτες περιέχουν διευθύνσεις, ο μεταγλωττιστής θα πρέπει να γνωρίζει τον τύπο που είναι αποθηκευμένος σε μια διεύθυνση έτσι ώστε να μπορεί να προσπελάσει την τιμή στην οποία δείχνει ο δείκτης.



```
void foo() {  
    int val;  
    int *valptr;  
    int **valptrptr;  
  
    val = 16;  
    valptr = &val;  
    *valptr = 5;  
  
    valptrptr = &valptr;  
    *valptrptr = NULL;  
}
```

# Δείκτες

- Καθώς και ένας δείκτη σε δείκτη αποθηκεύει μια διεύθυνση, μπορεί να χρησιμοποιηθεί σχεδόν όπως ένας δείκτης.
- Ο επιπλέον δείκτης απλά σημαίνει ότι η τιμή της διεύθυνσης είναι η διεύθυνση ενός `int*` και όχι ενός `int`.

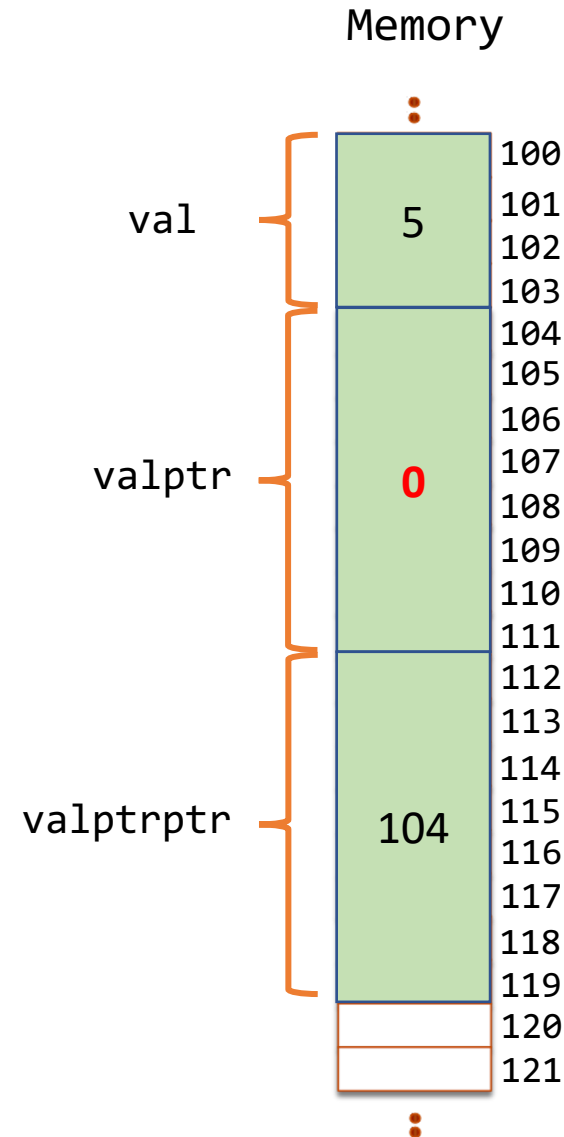


```
void foo() {  
    int val;  
    int *valptr;  
    int **valptrptr;  
  
    val = 16;  
    valptr = &val;  
    *valptr = 5;  
  
    valptrptr = &valptr;  
    *valptrptr = NULL;  
}
```

# Δείκτες

- Η ειδική θέση μνήμης NULL (0x0) είναι δεσμευμένη για να υποδηλώσει μνήμη που δεν μπορεί να χρησιμοποιηθεί.
- Προτείνεται οι δείκτες να αρχικοποιούνται στην τιμή NULL.

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample1.cpp>



```
void foo() {  
    int val;  
    int *valptr;  
    int **valptrptr;  
  
    val = 16;  
    valptr = &val;  
    *valptr = 5;  
  
    valptrptr = &valptr;  
    *valptrptr = NULL;  
}
```

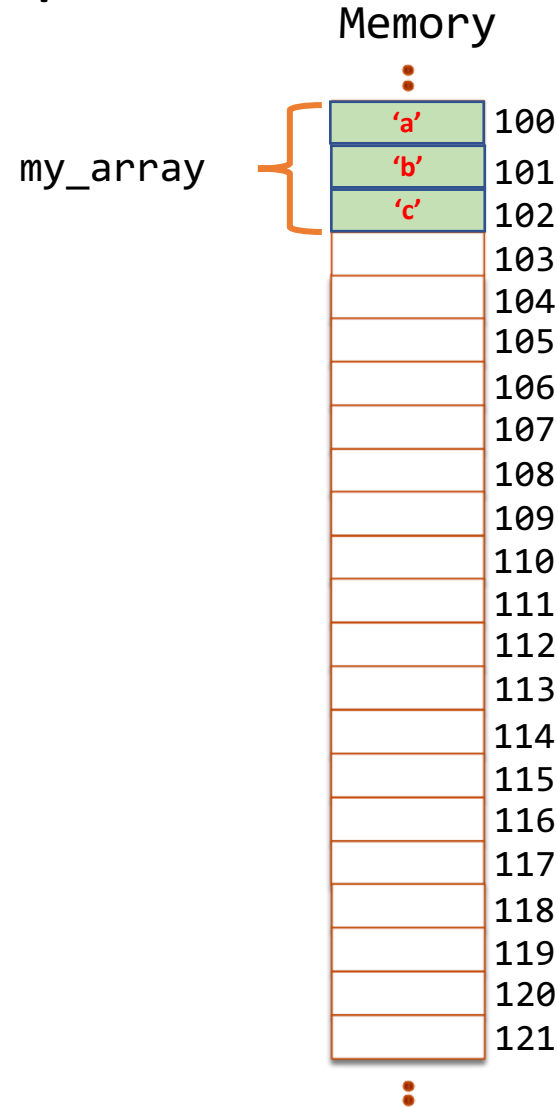
# Τοπικές μεταβλητές

- Οι τοπικές μεταβλητές καταλαμβάνουν χώρο στη στοίβα.
- Η διάρκεια ζωής μιας τοπικής μεταβλητής καθορίζεται από την συνάρτηση στην οποία δηλώνεται.
  - Η μεταβλητή δημιουργείται όταν καλείται η συνάρτηση
  - Όταν η συνάρτηση επιστρέφει, όλες οι τοπικές μεταβλητές παύουν να υπάρχουν (η δε μνήμη που καταλάμβαναν επιστρέφει στο σύστημα).
  - Προσοχή στη χρήση δεικτών που δείχνουν σε τοπικές μεταβλητές συναρτήσεων
    - Δεν θα πρέπει ποτέ μια συνάρτηση να επιστρέφει έναν δείκτη σε τοπική μεταβλητή.

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample2.cpp>

# Πίνακες και αριθμητική δεικτών

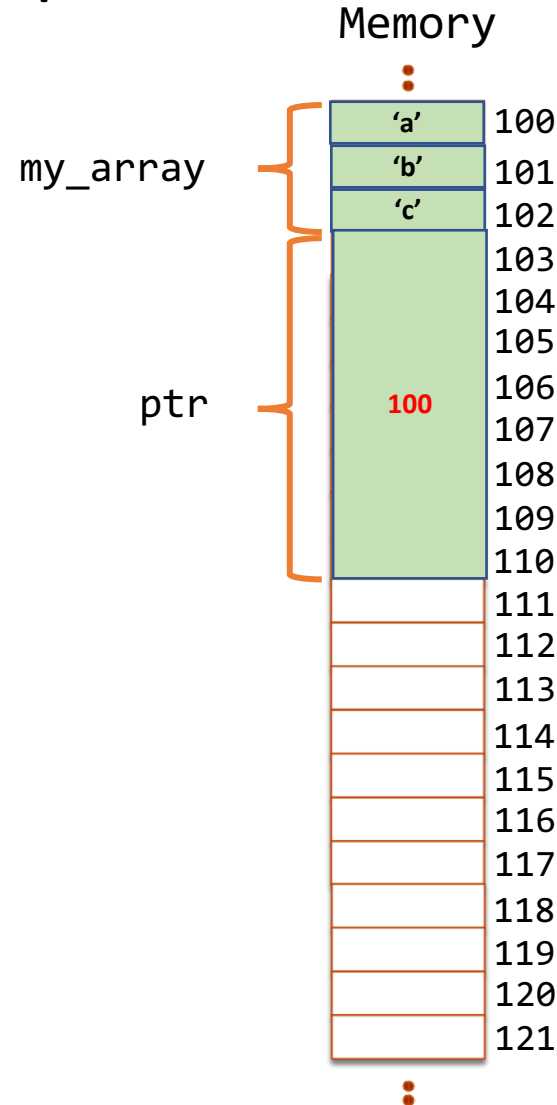
- Όταν ένας πίνακας δηλώνεται, ο μεταγλωττιστής δεσμεύει στη μνήμη το χώρο μνήμης που απαιτείται για όλα τα στοιχεία του πίνακα.
- Ο δείκτης λειτουργεί πλέον ως ένας δείκτης προς το πρώτο στοιχείο του πίνακα, αν και αυτό είναι μια αφαίρεση καθώς δεν υπάρχει αποθηκευμένος πραγματικά ένας δείκτης στη μνήμη.



```
void foo() {  
    char my_array[3] = {'a',  
        'b', 'c'};  
    char *ptr;  
    ptr = my_array;  
}
```

# Πίνακες και αριθμητική δεικτών

- Ένας «πραγματικός» δείκτης δεσμεύει μνήμη στην οποία μπορεί να αποθηκευτεί μια διεύθυνση μνήμης.
- Όταν ένας πίνακας αναφέρεται σε ένα πρόγραμμα χωρίς τον τελεστή `[]`, ο μεταγλωττιστής «θεωρεί» ότι το όνομα του πίνακα αναφέρεται σε έναν δείκτη που περιέχει τη διεύθυνση του πρώτου στοιχείου του πίνακα. Έτσι μπορεί να ανατεθεί σε έναν δείκτη ο πίνακας.



```
void foo() {  
    char my_array[3] = {'a',  
                        'b', 'c'};  
    char *ptr;  
    ptr = my_array;  
}
```

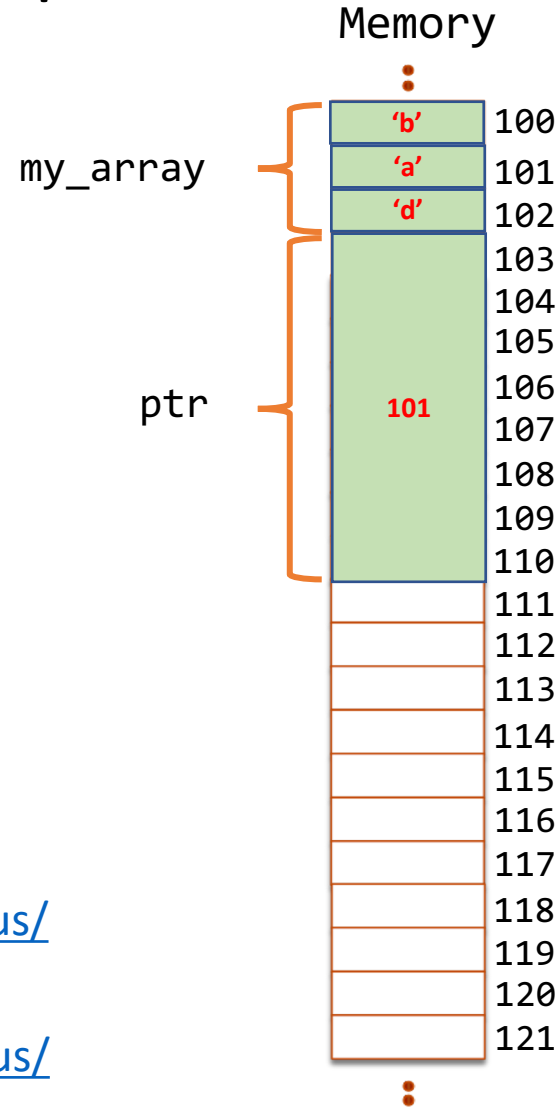


# Πίνακες και αριθμητική δεικτών

- Οι δείκτες μπορούν να τροποποιηθούν με τους τελεστές `+`, `-`, `++`, `--`. Μια βασική διαφορά μεταξύ ενός δείκτη και ενός ακεραίου είναι ότι η εντολή `ptr = ptr + num`, μεταφράζεται σε `ptr = ptr + num * (το μέγεθος του τύπου στον οποίο δείχνει ο δείκτης)`.
- Η αλλαγή της τιμής ενός δείκτη με αριθμητικές πράξεις είναι γνωστή ως αριθμητική δεικτών. Μεταξύ άλλων, είναι χρήσιμη για να διασχίσουμε έναν πίνακα.
- Η αριθμητική δεικτών και η αποαναφορά μπορούν να συνδυαστούν σε μια εντολή για να προσπελαστούν θέσεις σε διάφορες αποστάσεις από την αρχή του πίνακα. Όταν ο μεταγλωττιστής συναντά έναν δεικτοδοτημένο πίνακα, η εντολή μεταφράζεται από `array[index]` σε `*(array+index)`

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample3.cpp>

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample4.cpp>



```
void foo() {  
    char my_array[3] = {'a', 'b', 'c'};  
    char *ptr;  
    ptr = my_array;  
  
    ptr = ptr + 1;  
    *ptr = *my_array;  
    *(ptr+1) = 'd';  
    ptr[1] = 'b';  
}
```

# Επιπλέον πληροφορίες για τους δείκτες

- Ένας δείκτης μπορεί να ανατεθεί σε έναν άλλο δείκτη αν είναι του ίδιου τύπου ή αν γίνει μετατροπή τύπου (casting).
- Οποιοσδήποτε δείκτης μπορεί να ανατεθεί σε δείκτη `void*`.
- Η αποαναφορά ενός μη αρχικοποιημένου δείκτη ή ενός `NULL` δείκτη θα προκαλέσει σφάλμα κατάτμησης (segmentation fault).

<https://github.com/chgogos/oop/blob/master/variuous/COP3330/lect9/sample5.cpp>

# Αναφορές (references)

- Οι αναφορές είναι μια υποκατηγορία δεικτών που χρησιμοποιείται στη C++.
- Πρόκειται για ένα περιορισμένων δυνατοτήτων δείκτη που μπορεί να δείχνει προς ένα μόνο αντικείμενο ή μια μόνο μεταβλητή.
- Δηλώνεται χρησιμοποιώντας τον τελεστή & και όχι το \* (`int &ref`).
- Δεν μπορεί να υπάρχει αναφορά σε αναφορά.
- Μια αναφορά πρέπει να αρχικοποιείται κατά τη δήλωση της.
- Από τη στιγμή που αρχικοποιείται μια αναφορά δεν μπορεί να αλλάξει τιμή προς ένα άλλο αντικείμενο ή μια άλλη μεταβλητή.
- Δεν μπορεί να εφαρμοστεί αριθμητική δεικτών σε αναφορές.
- Οι αναφορές πραγματοποιούν αυτόματη αποαναφορά και συνεπώς μόνο η τιμή της μεταβλητής που αναφέρεται είναι προσβάσιμη.
- Μια αναφορά λειτουργεί ως δεύτερο όνομα ή ως ψευδώνυμο για μια μεταβλητή ή ένα αντικείμενο.

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample6.cpp>

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample7.cpp>

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample8.cpp>

<https://github.com/chgogos/oop/blob/master/various/COP3330/lect9/sample9.cpp>

# Πειραματισμός με δείκτες: pythontutor

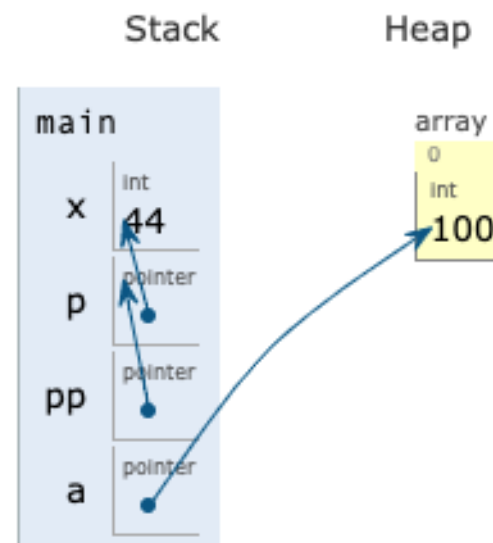
C++ (gcc 4.8, C++11)

EXPERIMENTAL! [known limitations](#)

```
1  #include <iostream>
2
3  int main() {
4      int x = 42;
5      int *p = &x;
6      *p = 43;
7      std::cout << p << " " << *p << std::endl;
8      int **pp = &p;
9      **pp = 44;
10     std::cout << *pp << " " << **pp << std::endl;
11     int *a = new int;
12     *a = 100;
13     std::cout << a << " " << *a << std::endl;
14     delete a;
15     return 0;
16 }
```

Print output (drag lower right corner to resize)

```
0xffff000bb4 43
0xffff000bb4 44
0x5c22040 100
```



<http://pythontutor.com/cpp.html>

# Πειραματισμός με δείκτες: skorbut

The screenshot displays the skorbut debugger interface. The top window shows the memory state for the `main` function. It includes variables `x` (value 52), `p` (pointer), `pp` (pointer), and an array `a` with elements [7, 2, 3, 4, 7]. Arrows indicate that `p` points to the first element of `a` and `pp` points to `p`. The right window shows the C code for `main`, with the `return 0;` statement highlighted. The bottom window shows the console output: `52` and `main finished with exit code 0`.

```
1 int main() {  
2     int x = 42;  
3     int *p = &x;  
4     int **pp = &p;  
5     **pp = 52;  
6     printf("%d\n", *p);  
7     int a[]={1,2,3,4,5};  
8     p = a;  
9     *p = 7;  
10    *(p+4) = 7;  
11    return 0;  
12 }  
13
```

52  
main finished with exit code 0

# Ερωτήσεις σύνοψης

- Τι αποθηκεύεται στο a όταν υπάρχει η δήλωση `int* a;`
- Τι αποθηκεύεται στο a όταν υπάρχει η δήλωση `int** a;`
- Γιατί δεν θα πρέπει να επιστρέφουμε έναν δείκτη προς μια τοπική μεταβλητή;
- Ποια είναι η μορφή με την οποία με δείκτη μπορούμε να αναφερθούμε στο στοιχείο `a[100]`;
- Τι σημαίνει ο δείκτης `NULL`;
- Ποια είναι η σχέση ανάμεσα σε δείκτες και αναφορές;
- Ποια είναι τα μεγέθη των `p1`, `p2` και `p3` στον ακόλουθο κώδικα;

```
int *p1;  
char *p2;  
Fraction *p3;
```

- Ποια είναι η έξοδος του ακόλουθου κώδικα;

```
int a = 5;  
int *ptr = &a;  
cout << ptr;  
cout << *ptr;
```

# Αναφορές

- <http://www.cs.fsu.edu/~xyuan/cop3330/>
- <http://pythontutor.com/cpp.html>
- <https://github.com/fredoverflow/skorbut-release>