Εισαγωγή στον Προγραμματισμό

Μέρος 6ο: Δείκτες

Εξάμηνο Σπουδών: 3ο

Κωδικός Μαθήματος: 343

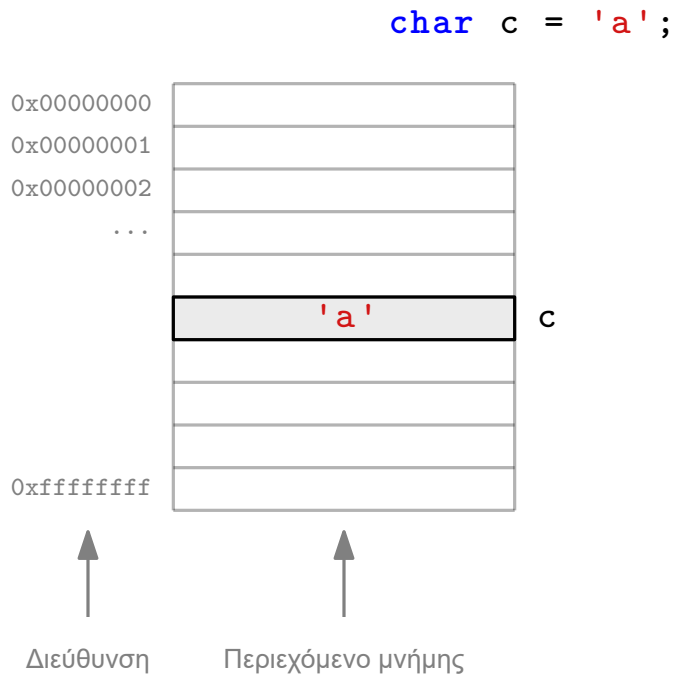
Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Μέρος 1°:
Εισαγωγή στους δείκτες

Μεταβλητές

- Μια **μεταβλητή** είναι ένα όνομα για ένα κομμάτι μνήμης που έχει μια τιμή (σ.σ., ένα μοναδικό αριθμό).
- Κατά την αρχικοποίηση μιας μεταβλητής, μια διεύθυνση ελεύθερης μνήμης εκχωρείται αυτόματα στη μεταβλητή.
- Οποιαδήποτε τιμή εκχωρούμε στη μεταβλητή αποθηκεύεται σε αυτή τη διεύθυνση μνήμης.
- 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 (64) bit Η/Υ η διεύθυνση είναι 32 (64) bits
- Η διεύθυνσης μιας μεταβλητής διαφέρει από το περιεχόμενο της



Μνήμη

- 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.

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

int main() {
  int i;
  char c;
  float f;
  double d;

  cout << "Size of char: "
       << sizeof(c) << endl;

  cout << "Size of int : "
       << sizeof(i) << endl;


  cout << "Size of float : "
       << sizeof(f) << endl;

  cout << "Size of double: "
       << sizeof(d) << endl;

  return 0;
}
```

Part06/sizeof.cpp

Οι τελεστές διεύθυνσης (&) και αποαναφοράς (*)

- Τελεστής διεύθυνσης (&) [address-of operator]:

  Επιστρέφει τη διεύθυνση της μνήμης μιας μεταβλητής.

- Τελεστής αποαναφοράς (*) [dereference operator]:

  Επιστρέφει την τιμή μιας διεύθυνσης μνήμης.

- **Παρατήρηση:** Ο τελεστής εισαγωγής << ερμηνεύει τις διευθύνσεις μνήμης ως δεκαεξαδικές

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

int main() {
  // declare an integer variable
  int x = 5;

  // output the value of variable x
  cout << "The value of x is: "
       << x << endl;

  // output the memory address of
  // variable x
  cout << "The address of x is: "
       << &x << endl;

  // output the value at the memory
  // address of variable x
  cout << "The value at address of x is: "
       << *&x << endl;

  return 0;
}
```

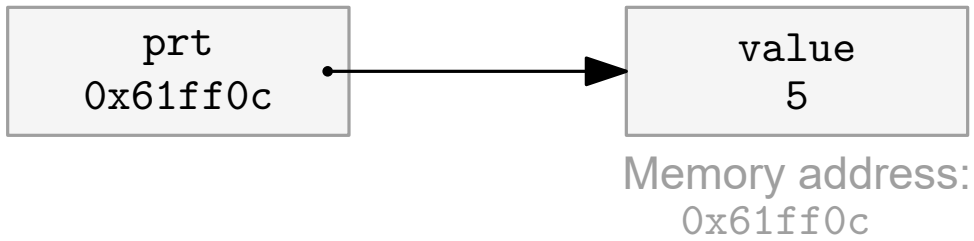
Part06/addressof.cpp

Δείκτες

- Ένας δείκτης είναι μια μεταβλητή που κρατά μια διεύθυνση μνήμης ως τιμή της.

- **Ανάθεση τιμής σε δείκτη:** Πριν χρησιμοποιηθεί ένας δείκτης, πρέπει να τοποθετηθεί σε αυτόν μια συγκεκριμένη διεύθυνση:

  - ```
    int value = 5;
    int *ptr = &value;
    ```



- Ο αστερίσκος σημαίνει “δείκτης προς”

```
  &x = 0x61ff0c
 ptr = 0x61ff0c
--------------
   x = 5
  &x = 0x61ff0c
 *&x = 5
 ptr = 0x61ff0c
*ptr = 5
&ptr = 0x61ff08
```

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

int main() {
  // declare a variable and a pointer
  // holding its address
  int x = 5;
  int *ptr = &x;

  // output the address of variable x
  cout << "  &x = " << &x << endl;

  // output the address that ptr holds
  cout << " ptr = " << ptr << endl;

  // more...
  cout << "---------------" << endl;
  cout << "   x = " <<    x << endl;
  cout << "  &x = " <<   &x << endl;
  cout << " *&x = " <<  *&x << endl;
  cout << " ptr = " <<  ptr << endl;
  cout << "*ptr = " << *ptr << endl;
  cout << "&ptr = " << &ptr << endl;
}
```

Part06/ptrintro.cpp
```

Δήλωση ενός δείκτη

- Η δήλωση ενός δείκτη γίνεται ως εξής:
  όνομα_τύπου *όνομα_δείκτη;

- Ο μεταγλωττιστής πρέπει να γνωρίζει σε τι είδους (τύπου) μεταβλητή δείχνει ο δείκτης

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

int main() {
  // a pointer to an integer value
  int *iPtr;

  // a pointer to a double value
  double *dPtr;

  // also valid (acceptable; not favored)
  int* iPtr2;

  // also valid (but do not do this!)
  int * iPtr3;

  // two pointers to string objects
  string *iPtr4, *iPtr5;

  return 0;
}
```

Part06/ptrdecl.cpp

Δήλωση ενός δείκτη

- Η δήλωση ενός δείκτη γίνεται ως εξής:
  όνομα_τύπου *όνομα_δείκτη;
- Ο μεταγλωττιστής πρέπει να γνωρίζει σε τι είδους (τύπου) μεταβλητή δείχνει ο δείκτης
- Ο τύπος ενός δείκτη πρέπει να ταιριάζει με τον τύπο της μεταβλητής στην οποία δείχνει.

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

int main() {
  // declare two variables (int & double)
  int iValue = 5;
  double dValue = 7.0;

  // proper statements
  int *iPtr = &iValue;
  double *dPtr = &dValue;

  // wrong: int pointer cannot point to
  // the address of a double variable
  iPtr = &dValue;

  // wrong: double pointer cannot point
  // to the address of an int variable
  dPtr = &iValue;

  return 0;
}
```

Part06/ptrtype.cpp

Πρόσβαση στη μεταβλητή

- Σύνταξη για την πρόσβαση στην τιμή μιας μεταβλητής με τη χρήση δείκτη (αντί του ονόματός της):

  ***όνομα_δείκτη**

- Τελεστής αποαναφοράς δείκτη: Ένας αστερίσκος μπροστά από το όνομα ενός δείκτη.
  - Π.χ., `int pointedVar = *ptr;`
  - Επιστρέφει την τιμή της μεταβλητής που δείχνει
  - Χρησιμοποιείται και για την αλλαγή της τιμής του περιεχομένου της διεύθυνσης που κρατά
  - Είναι διαφορετικός από τον τελεστή-αστερίσκο για τη δήλωση δείκτη.

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

int main() {
  // an integer variable x
  int  x = 27;

  // a pointer to an integer variable
  // initialized to the address of x
  int  *ptr = &x;

  // print the integer variable, the
  // address the pointer holds and the
  // content of the pointer
  cout << "   x = " <<    x << endl;
  cout << " ptr = " <<  ptr << endl;
  cout << "*ptr = " << *ptr << endl;

  // change content of pointer and print
  *ptr = 47;
  cout << "   x = " <<    x << endl;
  cout << " ptr = " <<  ptr << endl;
  cout << "*ptr = " << *ptr << endl;
}
```

Part06/ptrvalue.cpp

Σύνοψη

- Ένας δείκτης μπορεί να κρατήσει τη διεύθυνση οποιασδήποτε μεταβλητής.
  - είναι ένα υποδοχέας που περιμένει μια διεύθυνση
- Ό τύπος του δείκτη θα πρέπει να συμφωνεί με τον τύπο της μεταβλητής που κρατά.
- Κάθε δείκτης πρέπει να έχει μια τιμή.
  - αλλιώς θα δείχνει σε μια διεύθυνση που δεν θέλουμε να δείχνει
- Βεβαιωθείτε ότι αποδίδετε σε κάθε δείκτη μια έγκυρη διεύθυνση πριν πριν τη χρήση του.
  - λανθασμένες τιμές δεικτών μπορούν να οδηγήσουν σε κατάρρευση του συστήματος του Η/Υ

Παραδείγματα

- Τι εκτυπώνει το παρακάτω πρόγραμμα;

```
int  x = 0;
int *p = &x;
*p = 42;
cout <<  x << endl;
cout << *p << endl;
```

- Τι εκτυπώνει το παρακάτω πρόγραμμα;

```
int x = 0;
int *p1, *p2;
p1 = &x;
p2 = p1;
x = 10;
cout << *p1 << endl;
cout << *p2 << endl;
```

Παραδείγματα

- Τι εκτυπώνει το παρακάτω πρόγραμμα;

```
int  x = 0;
int *p = &x;
*p = 42;
cout <<  x << endl;
cout << *p << endl;
```

```
42
42
```

- Τι εκτυπώνει το παρακάτω πρόγραμμα;

```
int x = 0;
int *p1, *p2;
p1 = &x;
p2 = p1;
x = 10;
cout << *p1 << endl;
cout << *p2 << endl;
```

```
10
10
```

Δείκτες ως ορίσματα συναρτήσεων

- Ένας δείκτης μπορεί να είναι όρισμα σε μια συνάρτηση.

- Στην περίπτωση αυτή, ο δείκτης λειτουργεί ως παράμετρο κλήσης αναφοράς.
  - Αν θέλουμε να αποτρέψουμε μια τέτοια συμπεριφορά θα πρέπει να ορίσουμε τον δείκτη const στη δήλωση της συνάρτησης.

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

void swap(int &x, int &y);
void swap(int *x, int *y);

int main() {
  int x = 1, y = 2;
  cout << "x:" << x << ",y:" << y <<endl;

  swap(x, y);  //swap (by reference)
  cout << "x:" << x << ",y:" << y <<endl;

  swap(&x, &y);//swap back with pointers
  cout << "x:" << x << ",y:" << y <<endl;
}

void swap(int &x, int &y) {
  int temp = x; x = y; y = temp;
}

void swap(int *x, int *y) {
  int temp = *x; *x = *y; *y = temp;
}
```

Part06/ptrfunc.cpp

Δείκτες σε αντικείμενα

- Ένας δείκτης σε αντικείμενο αποθηκεύει τη διεύθυνση ενός αντικειμένου μιας κλάσης.
- Μπορεί να χρησιμοποιηθεί για την πρόσβαση στα μέλη (πεδία, μέθοδοι) της κλάσης.
- **Σύνταξη:**

1. Δημιουργία αντικείμενου:

```
ClassName objectName(parameters);
```

2. Δημιουργία δείκτη προς στο αντικείμενο:

```
ClassName *pointerName = &objectName;
```

3. Κλήση μεθόδων:

```
(*pointerName).functionName(parameters);
```

Δείκτες σε αντικείμενα

- Ένας δείκτης σε αντικείμενο αποθηκεύει τη διεύθυνση ενός αντικειμένου μιας κλάσης.
- Μπορεί να χρησιμοποιηθεί για την πρόσβαση στα μέλη (πεδία, μέθοδοι) της κλάσης.
- **Σύνταξη:**

1. Δημιουργία αντικείμενου:

```
ClassName objectName(parameters);
```

2. Δημιουργία δείκτη προς στο αντικείμενο:

```
ClassName *pointerName = &objectName;
```

Ισοδύναμα:

```
ClassName *objectName = new ClassName(parameters);
```

3. Κλήση μεθόδων:

```
(*pointerName).functionName(parameters);
```

Δείκτες σε αντικείμενα

- Ένας δείκτης σε αντικείμενο αποθηκεύει τη διεύθυνση ενός αντικειμένου μιας κλάσης.
- Μπορεί να χρησιμοποιηθεί για την πρόσβαση στα μέλη (πεδία, μέθοδοι) της κλάσης.
- **Σύνταξη:**

1. Δημιουργία αντικείμενου:

```
ClassName objectName(parameters);
```

2. Δημιουργία δείκτη προς στο αντικείμενο:

```
ClassName *pointerName = &objectName;
```

Ισοδύναμα:

```
ClassName *objectName = new ClassName(parameters);
```

3. Κλήση μεθόδων:

```
(*pointerName).functionName(parameters);
```

Ισοδύναμα:

```
pointerName->functionName(parameters);
```

Δείκτες σε αντικείμενα

- Ένας δείκτης σε αντικείμενο αποθηκεύει τη διεύθυνση ενός αντικειμένου μιας κλάσης.
- Μπορεί να χρησιμοποιηθεί για την πρόσβαση στα μέλη (πεδία, μέθοδοι) της κλάσης.
- **Σύνταξη:**

1. Δήλωση δείκτη:

```
ClassName *objectName;
```

2. Αρχικοποίηση:

```
objectName = new ClassName(parameters);
```

3. Κλήση μεθόδων:

```
pointerName->functionName(parameters);
```

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

class Circle {
private:
  int radius;
public:
  Circle() : radius(1) {}
  double area() {
    return 3.14*radius*radius;
  }
  double circumference() {
    return 2*3.14*radius;
  }
};
int main() {
  // Define a named circle object
  Circle c;
  cout << "Area: " << c.area() << endl;

  // Pointer to a circle object
  Circle *cptr = new Circle();
  cout << "Area: " << cptr->area() << endl;
}
```

Part06/ptrobj.cpp

Δείκτες και πίνακες

- Στη C++ το όνομα ενός πίνακα είναι ένας **σταθερός** δείκτης στο πρώτο στοιχείο του πίνακα.

- Στο παρακάτω παράδειγμα η μεταβλητή `x` είναι ένας δείκτης στην πρώτη θέση του πίνακα:
  - `int x[5] = {0,1,2,3,4};`


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

const int SIZE = 5;

int main() {
  int x[SIZE] = {0, 1, 2, 3, 4};

  // pointer to the base address of x
  // equivalently: int *ptr = &x[0];
  int *ptr = x;

  // use the pointer to access array x
  for (int i=0; i<SIZE; i++) {
    cout << *(ptr + i) << " ";
  }
  cout << endl;

  // since x is a pointer, equivalently
  for (int i=0; i<SIZE; i++) {
    cout << *(x + i) << " ";
  }
}
```

Part06/ptrarray.cpp

Δείκτες και πίνακες

- Στη C++ το όνομα ενός πίνακα είναι ένας **σταθερός** δείκτης στο πρώτο στοιχείο του πίνακα.

- Στο παρακάτω παράδειγμα η μεταβλητή `x` είναι ένας δείκτης στην πρώτη θέση του πίνακα:
  - `int x[5] = {0,1,2,3,4};`

- Σύνηθες σφάλμα:

```
int x[5] = {0,1,2,3,4};
int *ptr = &x[0];
ptr++;  // valid increment
x++   // invalid increment
```

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

const int SIZE = 5;

int main() {
  int x[SIZE] = {0, 1, 2, 3, 4};

  // pointer to the base address of x
  // equivalently: int *ptr = &x[0];
  int *ptr = x;

  // use the pointer to access array x
  for (int i=0; i<SIZE; i++) {
    cout << *(ptr + i) << " ";
  }
  cout << endl;

  // since x is a pointer, equivalently
  for (int i=0; i<SIZE; i++) {
    cout << *(x + i) << " ";
  }
}
```

Part06/ptrarray.cpp

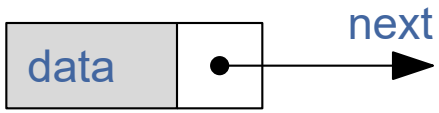
Μέρος 2°:

Εφαρμογή: Συνδεδεμένες λίστες

Λίστα

- Η λίστα είναι μια δομή δεδομένων, η οποία ορίζεται ως μια ακολουθία κόμβων.

- Κάθε κόμβος αποθηκεύει:
  - ένα στοιχείο
  - ένα δείκτη στον επόμενο κόμβο



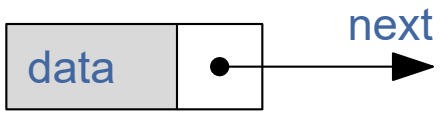

```
/**
 * A node in a list maintains:
 * - an integer value
 * - a pointer to a next node (if any).
 */
#include <cstdlib> // for NULL

class Node {
public:
  int data;
  Node* next;

  // Default constructor
  Node() {
    data = 0;
    next = NULL;
  }

  // Parameterised Constructor
  Node(int d) {
    data = d;
    next = NULL;
  }
};
```

Part06/list/Node.cpp

Λίστα

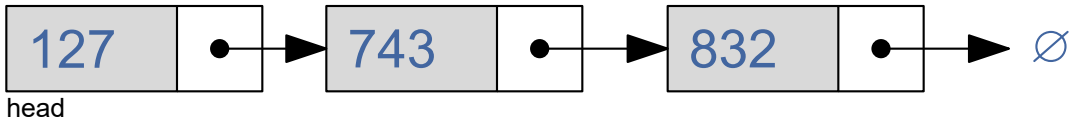
- Η λίστα είναι μια δομή δεδομένων, η οποία ορίζεται ως μια ακολουθία κόμβων.

- Κάθε κόμβος αποθηκεύει:
  - ένα στοιχείο
  - ένα δείκτη στον επόμενο κόμβο



- Λίστα $\rightarrow$ Δείκτης (head) στον πρώτο κόμβο.

- Παράδειγμα λίστας:




```cpp
#include <cstdlib> // for NULL
#include "Node.cpp"

class Linkedlist {
private:
  Node* head;  //The first node

public:
  // Default constructor
  Linkedlist() { head = NULL; }

  // Returns the length of the list
  int size() const;

  // Prints all elements
  void print() const;

  // Inserts an element as last
  void push_back(const int element);

  // Deletes the element at position
  void erase(const int position);
};
```

Part06/list/Linkedlist.hpp

Διαπέραση μιας λίστας

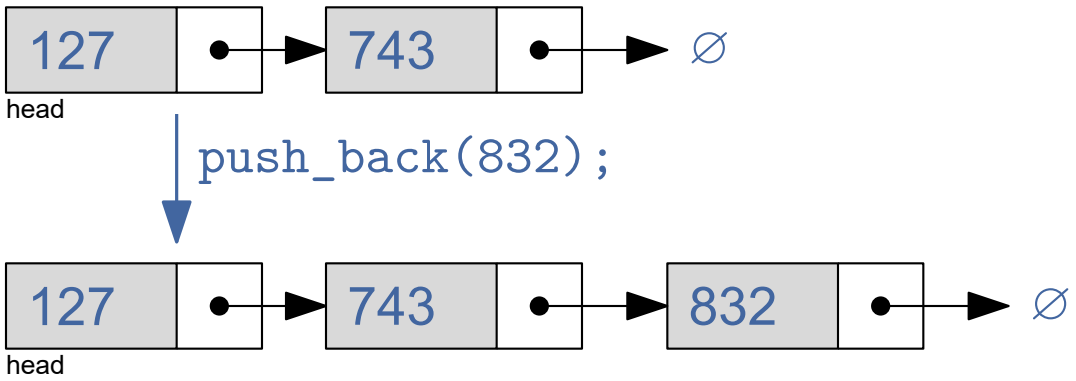
- Η διαπέραση μιας λίστας γίνεται διατηρώντας ένα δείκτη ο οποίος δείχνει αρχικά στον πρώτο κόμβο της λίστας.
- Σε κάθε επανάληψη, αφού ολοκληρώσουμε την επίσκεψη στον κόμβο, μεταφέρουμε τον δείκτη στον επόμενο κόμβο.
- Η διαπέραση ολοκληρώνεται όταν επισκεφτούμε ένα κόμβο `NULL`.

```
// Returns the length of the list
int Linkedlist::size() const {
  Node *node = head;
  int size = 0;

  // traverse the list.
  while (node != NULL) {
    size++;
    node = node->next;
  }
  return size;
}

// Prints all elements
void Linkedlist::print() const {
  Node *node = head;

  // traverse the list.
  while (node != NULL) {
    cout << node->data << " ";
    node = node->next;
  }
  cout << endl;
}
```

Part06/list/Linkedlist.cpp

Εισαγωγή στοιχείου στη λίστα

- Η εισαγωγή ενός νέου στοιχείου στη λίστα, γίνεται δημιουργώντας ένα νέο κόμβο.
- Αν η λίστα είναι κενή, ο κόμβος αυτός γίνεται ο πρώτος (head) της λίστας.
- Διαφορετικά, διαπερνάμε ολόκληρη τη λίστα και τον εισάγουμε ως τελευταίο (δηλαδή, ως επόμενο του τελευταίου κόμβου).

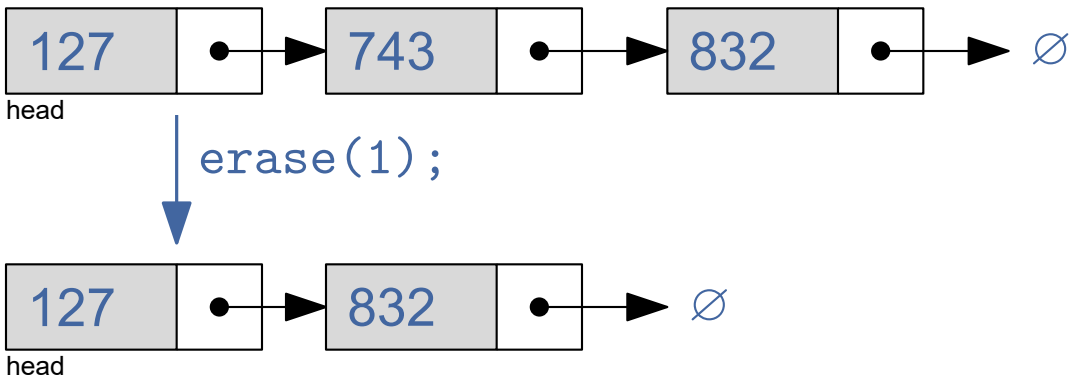



```
/**
 * Inserts an element as last in the list
 */
void Linkedlist::push_back(const int data)
{
  // Create the new Node.
  Node *newNode = new Node(data);

  // Assign to head
  if (head == NULL) {
    head = newNode;
    return;
  }

  // Traverse till end of list
  Node *temp = head;
  while (temp->next != NULL) {
    // Update temp
    temp = temp->next;
  }

  // Insert at the last.
  temp->next = newNode;
}
```

Part06/list/Linkedlist.cpp

Διαγραφή στοιχείου

- Η διαγραφή του στοιχείου από συγκεκριμένη θέση της λίστας, γίνεται εντοπίζοντας τον κόμβο στη θέση αυτή και τον προηγούμενο κόμβο.
- Έστω `node` ο πρώτος από τους δύο αυτούς κόμβους και `previous` ο δεύτερος.
- Η διαγραφή γίνεται ορίζοντας ως επόμενο κόμβο του `previous` τον επόμενο κόμβο του κόμβου `node`.




```
/**
 * Deletes the element at given index
 */
void Linkedlist::erase(const int index) {
  // Empty list or index out of range
  if (head==NULL || size() < index) {
    return;
  }

  // Deleting the head
  if (size() == 1) {
    head = head->next;
    return;
  }

  // Find the node to be deleted.
  Node *node = head, *previous = NULL;
  for (int i=0; i<index; i++) {
    previous = node;
    node = node->next;
  }
  // Delete by excanging pointers
  previous->next = node->next;
}
```

Part06/list/Linkedlist.cpp

Επιπλέον υλικό

- Κεφάλαιο 10 (σ.σ., 456–499):
  R. Lafore, Αντικειμενοστρεφής προγραμματισμός με τη C++,
  ISBN: 960-209-904-6, εκδόσεις Κλειδάριθμος, 2006.