Πανεπιστήμιο Πατρών

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

ΟΝΤΟΚΕΝΤΡΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΙΙ (C++)

Δείκτες και Συμβολοσειρές (Pointers & Strings)

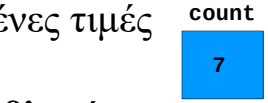
Περιεχόμενα

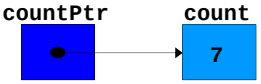
Εισαγωγή

- Δείκτες (Pointers)
  - Ισχυρό χαρακτηριστικό, αλλά δύσκολη η διαχείρισή τους
  - Υλοποιούν ένα είδος pass-by-reference
  - Στενή σχέση με τους πίνακες και τα strings

Δήλωση και αρχικοποίηση μεταβλητών τύπου δείκτη

- Μεταβλητές τύπου δείκτη
  - Αποθηκεύουν διευθύνσεις μνήμης ως τιμές
  - Μια κανονική μεταβλητή αποθηκεύει συγκεκριμένες τιμές (άμεση αναφορά - direct reference)
  - Ένας δείκτης αποθηκεύει τη διεύθυνση μιας μεταβλητής που περιέχει μια συγκεκριμένη τιμή (έμμεση αναφορά - indirect reference)





- Έμμεση αναφορά (Indirection)
  - Η αναφορά μιας τιμής μέσω ενός δείκτη
- Δήλωση δείκτη
  - Το * στη δήλωση υποδηλώνει ότι η μεταβλητή είναι δείκτης

```
int *myPtr;
```

  - Περισσότεροι δείκτες χρειάζονται και περισσότερα *

```
int *myPtr1, *myPtr2;
```

Δήλωση και αρχικοποίηση μεταβλητών τύπου δείκτη

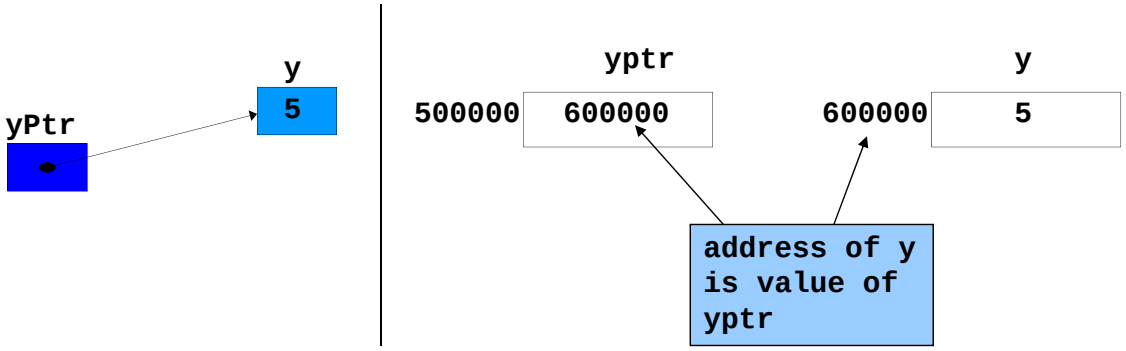
- Οι δείκτες μπορούν να χρησιμοποιηθούν με οποιοδήποτε τύπο δεδομένων.
- Αρχικοποίηση δείκτη
  - Αρχικοποιείται στο **`0`** ή **`NULL`**
    - **`0`** ή **`NULL`** δείχνουν στο κενό
    - Η συμβολική σταθερά NULL ορίζεται στο αρχείο επικεφαλίδας **`<iostream>`**

Τελεστές δεικτών

- **&** (τελεστής διεύθυνσης)
  - Επιστρέφει τη διεύθυνση μνήμης του τελεστέου
  - Παράδειγμα

```
int y = 5;
int *yPtr;
yPtr = &y;    // yPtr gets address of y
```

  - **`yPtr`** "points to" **`y`**



Τελεστές δεικτών

- * (τελεστής έμμεσης αναφοράς ή αποαναφοράς)
  - Επιστρέφει το αντικείμενο που δείχνει ο δείκτης
  - ***yPtr** επιστρέφει το **y** (αφού **yPtr** δείχνει στο **y**).
  - Ένας αποαναφοροποιημένος δείκτης είναι ένα lvalue

```
*yptr = 9;       // assigns 9 to y
```

- * και & είναι αντίστροφοι τελεστές (αλληλοαναιρούνται)

Outline

fig05_04.cpp (1 of 2)

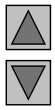
```
1  // Fig. 5.4: fig05_04.cpp
2  // Using the & and * operators.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int a;      // a is an integer
11    int *aPtr;  // aPtr is a pointer to an integer
12
13    a = 7;
14    aPtr = &a;  // aPtr assigned address of a
15
16    cout << "The address of a is " << &a
17         << "\nThe value of aPtr is " << aPtr;
18
19    cout << "\n\nThe value of a is " << a
20         << "\nThe value of *aPtr is " << *aPtr;
21
22    cout << "\n\nShowing that * and & are inverses of "
23         << "each other.\n&*aPtr = " << &*aPtr
24         << "\n*&aPtr = " << *&aPtr << endl;
25
```

***** and **&** are inverses of each other

```
26    return 0;  // indicates successful termination
27
28 } // end main
```

```
The address of a is 0012FED4
The value of aPtr is 0012FED4

The value of a is 7
The value of *aPtr is 7

Showing that * and & are inverses of each other.
&*aPtr = 0012FED4
*&aPtr = 0012FED4
```

***** and **&** are inverses; same result when both applied to **aPtr**



Outline

fig05_04.cpp (2 of 2)

fig05_04.cpp output (1 of 1)

Κλήση συναρτήσεων και πέρασμα ορισμάτων με αναφορά

- 3 τρόποι να περαστούν τα ορίσματα σε μια συνάρτηση
  - Pass-by-value
  - Pass-by-reference with reference arguments
  - Pass-by-reference with pointer arguments
- Με `return` μπορούμε να επιστρέψουμε μια μόνο τιμή από τη συνάρτηση
- Με τα ορίσματα που περνούν με αναφορά
  - είναι δυνατή η αλλαγή των αρχικών τιμών των ορισμάτων
  - είναι δυνατή η επιστροφή από μια συνάρτηση περισσότερων τιμών

Κλήση συναρτήσεων και πέρασμα ορισμάτων με αναφορά

- Pass-by-reference με ορίσματα δείκτες
  - Περνάμε τη διεύθυνση του ορίσματος χρησιμοποιώντας τον τελεστή διεύθυνσης **&**
  - Οι πίνακες δεν χρειάζονται τον τελεστή **&** αφού το όνομα του πίνακα αντιστοιχεί ήδη σε δείκτη
  - Χρησιμοποιούμε τον τελεστή έμμεσης αναφοράς ***** για την προσπέλαση των τιμών των μεταβλητών εντός της συνάρτησης

Outline

fig05_06.cpp (1 of 2)

```
1  // Fig. 5.6: fig05_06.cpp
2  // Cube a variable using pass-by-value.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int cubeByValue( int );   // prototype
9
10 int main()
11 {
12    int number = 5;
13
14    cout << "The original value of number is "
15
16    // pass number by value to cubeByValue
17    number = cubeByValue( number );
18
19    cout << "\nThe new value of number is " << number << endl;
20
21    return 0;  // indicates successful termination
22
23 } // end main
24
```

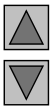
Pass number by value; result **returned** by **cubeByValue**

```
25 // calculate and return cube of integer argument
26 int cubeByValue( int n )
27 {
28    return n * n * n; // cube local va
29
30 } // end function cubeByValue
```

cubeByValue receives parameter passed-by-value

Cubes and **return**s local variable **n**

```
The original value of number is 5
The new value of number is 125
```



Outline

fig05_06.cpp (2 of 2)

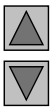
fig05_06.cpp output (1 of 1)


```
1  // Fig. 5.7: fig05_07.cpp
2  // Cube a variable using pass-by-reference
3  // with a pointer argument.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  void cubeByReference( int * );   // prototype
10
11 int main()
12 {
13    int number = 5;
14
15    cout << "The original value of numbe
16
17    // pass address of number to cubeByR
18    cubeByReference( &number );
19
20    cout << "\nThe new value of number is " << number << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
25
```

Prototype indicates parameter is pointer to **int**

Apply address operator **&** to pass address of number to **cubeByReference**

cubeByReference modified variable **number**



Outline

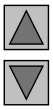
fig05_07.cpp (1 of 2)

```
26 // calculate cube of *nPtr; modifies variable number in main
27 void cubeByReference( int *nPtr )
28 {
29    *nPtr = *nPtr * *nPtr * *nPtr;  // cube
30
31 } // end function cubeByReference
```

```
The original value of number is 5
The new value of number is 125
```

cubeByReference receives address of **int** variable, i.e., pointer to an **int**

Modify and access **int** variable using indirection operator ***



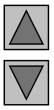
Outline

fig05_07.cpp (2 of 2)

fig05_07.cpp output (1 of 1)

Χρήση του const με δείκτες

- **const** qualifier
  - Δεν επιτρέπει την αλλαγή της τιμής της μεταβλητής
  - Βάζουμε το **const** όταν η συνάρτηση δεν χρειάζεται να αλλάξει την τιμή της μεταβλητής
- Αρχή του ελαχίστου δικαιώματος
  - Μια συνάρτηση δεν χρειάζεται να αποκτά παραπάνω δικαιώματα από αυτά που χρειάζεται για την επίτευξη της εργασίας της
- 4 τρόποι για να περάσουμε ένα δείκτη σε μια συνάρτηση
  - Μεταβλητός δείκτης σε μεταβλητά δεδομένα
    - Μέγιστο δικαίωμα προσπέλασης
  - Μεταβλητός δείκτης σε σταθερά δεδομένα
  - Σταθερός δείκτης σε μεταβλητά δεδομένα
  - Σταθερός δείκτης σε σταθερά δεδομένα
    - Ελάχιστο δικαίωμα προσπέλασης



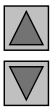
Outline

fig05_10.cpp (1 of 2)

```
1  // Fig. 5.10: fig05_10.cpp
2  // Converting lowercase letters to uppercase letters
3  // using a non-constant pointer to non-constant data.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <cctype>    // prototypes for islo
10
11 void convertToUppercase( char * );
12
13 int main()
14 {
15    char phrase[] = "characters and $32.98";
16
17    cout << "The phrase before conversion is: " << phrase;
18    convertToUppercase( phrase );
19    cout << "\nThe phrase after conversion is:  "
20         << phrase << endl;
21
22    return 0;  // indicates successful termination
23
24 } // end main
25
```

Parameter is nonconstant pointer to nonconstant data

convertToUppercase modifies variable phrase



Outline

fig05_10.cpp (2 of 2)

fig05_10.cpp output (1 of 1)

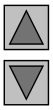
```
26 // convert string to uppercase letters
27 void convertToUppercase( char *sPtr )
28 {
29    while ( *sPtr != '\0' ) {   // current character is not '\0'
30
31       if ( islower( *sPtr ) )  // if character i
32          *sPtr = toupper( *sPtr );  // convert to uppercase
33
34       ++sPtr;  // move sPt
35
36    } // end while
37
38 } // end function convertTo

The phrase before conversion i
The phrase after conversion is
```

Parameter **sPtr** nonconstant pointer to nonconstant data

Function **islower** returns **true** if character is lowercase

Function **toupper** returns

When operator **++** applied to pointer that points to array, memory address stored in pointer modified to point to next element of array.



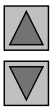
Outline

fig05_11.cpp (1 of 2)

```
1  // Fig. 5.11: fig05_11.cpp
2  // Printing a string one character at a time using
3  // a non-constant pointer to constant data.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  void printCharacters( const char * );
10
11 int main()
12 {
13    char phrase[] = "print characters of a s
14
15    cout << "The string is:\n";
16    printCharacters( phrase );
17    cout << endl;
18
19    return 0;  // indicates successful termination
20
21 } // end main
22
```

Parameter is nonconstant pointer to constant data.

Pass pointer **phrase** to function **printCharacters**.



Outline

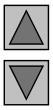
fig05_11.cpp (2 of 2)

fig05_11.cpp output (1 of 1)

```
23 // sPtr cannot modify the character to which it points,
24 // i.e., sPtr is a "read-only" pointer
25 void printCharacters( const char *sPtr )
26 {
27    for ( ; *sPtr != '\0'; sPtr++ )    // no initialization
28       cout << *sPtr;
29
30 } // end function printCharacters

The string is:
print characters of a string
```

sPtr is nonconstant pointer to constant data; cannot modify character to which

Increment **sPtr** to point to next character.



Outline

fig05_12.cpp (1 of 1)

fig05_12.cpp output (1 of 1)

```
1  // Fig. 5.12: fig05_12.cpp
2  // Attempting to modify data through a
3  // non-constant pointer to constant data.
4
5  void f( const int * );  // prototype
6
7  int main()
8  {
9     int y;
10
11    f( &y );   // f attempts illegal modification
12
13    return 0;  // indicates s
14
15 } // end main
16
17 // xPtr cannot modify the va
18 // to which it points
19 void f( const int *xPtr )
20 {
21    *xPtr = 100;  // error: cannot modify a const object
22
23 } // end function f
```

Parameter is nonconstant pointer to constant data.

Pass address of **int** variable **y** to attempt illegal modification.

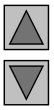
Attempt to modify **const** object pointed to by **xPtr**.

Error produced when attempting to compile.

```
d:\cpphtp4_examples\ch05\Fig05_12.cpp(21) : error C2166:
   l-value specifies const object
```


Σταθεροί δείκτες

- **const** pointers
  - Πάντα δείχνουν στην ίδια θέση μνήμης
  - Η περίπτωση του ονόματος ενός πίνακα
  - Πρέπει να αρχικοποιείται όταν δηλώνεται



Outline

fig05_13.cpp (1 of 1)

fig05_13.cpp output (1 of 1)

```
1  // Fig. 5.13: fig05_13.cpp
2  // Attempting to modify a constant pointer to
3  // non-constant data.
4
5  int main()
6  {
7     int x, y;
8
9     // ptr is a constant pointer to an int
10    // be modified through pt
11    // same memory location.
12    int * const ptr = &x;
13
14    *ptr = 7;  // allowed: *p
15    ptr = &y;  // error: ptr is const; can
16
17    return 0;  // indicates successful ter
18
19 } // end main
```

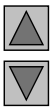
ptr is constant pointer to

Can modify **x** (pointed to by

Cannot modify **ptr** to point to new address since **ptr** is constant.

Line 15 generates compiler error by attempting to assign new address to constant pointer.

```
d:\cpphtp4_examples\ch05\Fig05_13.cpp(15) : error C2166:
   l-value specifies const object
```

Outline

fig05_14.cpp (1 of 1)

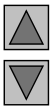
```
1  // Fig. 5.14: fig05_14.cpp
2  // Attempting to modify a constant pointer to constant data.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 5, y;
11
12    // ptr is a constant pointer to a co
13    // ptr always points to the same loc
14    // at that location cannot be modified.
15    const int *const ptr = &x
16
17    cout << *ptr << endl;
18
19    *ptr = 7;  // error: *ptr
20    ptr = &y;  // error: ptr is const; cannot assign new address
21
22    return 0;  // indicates successful termination
23
24 } // end main
```

ptr is constant pointer to integer constant.

Cannot modify **x** (pointed to

Cannot modify **ptr** to point to new address since **ptr** is constant.

```
d:\cpphtp4_examples\ch05\Fig05_14.cpp(19) : error C2166:
   l-value specifies const object
d:\cpphtp4_examples\ch05\Fig05_14.cpp(20) : error C2166:
   l-value specifies const object
```



Outline

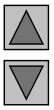
Line 19 generates compiler

Line 20 generates compiler error by attempting to assign new address to constant pointer.

_14.cpp
ut (1 of 1)

Σχέση δεικτών και πινάκων

- Πίνακες και δείκτες συνδέονται στενά
  - Το όνομα ενός πίνακα είναι ένας σταθερός δείκτης
  - Οι δείκτες μπορούν να πραγματοποιήσουν την προσπέλαση των στοιχείων ενός πίνακα
- Προσπέλαση στοιχείων πίνακα με δείκτες
  - Το στοιχείο **`b[ n ]`** μπορεί να προσπελαστεί με την έκφραση **`*( bPtr + n )`**
  - Διεύθυνση
    - **`&b[ 3 ]`** ίδιο με **`bPtr + 3`**
  - Το όνομα ενός πίνακα μπορεί να χρησιμοποιηθεί ως δείκτης
    - **`b[ 3 ]`** ίδιο με **`*( b + 3 )`**
  - Οι δείκτες μπορούν να χρησιμοποιηθούν με δείκτες
    - **`bPtr[ 3 ]`** ίδιο με **`b[ 3 ]`**

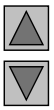


Outline

fig05_20.cpp (1 of 2)

```
1  // Fig. 5.20: fig05_20.cpp
2  // Using subscripting and pointer notations with arrays.
3
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int main()
10 {
11    int b[] = { 10, 20, 30, 40 };
12    int *bPtr = b;   // set bPtr to point to array b
13
14    // output array b using array subscript notation
15    cout << "Array b printed with:\n"
16         << "Array subscript notation\n";
17
18    for ( int i = 0; i < 4; i++ )
19       cout << "b[" << i << "] = " << b[ i ] << '\n';
20
21    // output array b using the array name and
22    // pointer/offset notation
23    cout << "\nPointer/offset notation where "
24         << "the pointer is the array name\n";
25
```

Using array subscript notation.



Outline

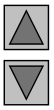
fig05_20.cpp (2 of 2)

```
26    for ( int offset1 = 0; offset1 < 4; offset1++ )
27       cout << "*(b + " << offset1 << ") = "
28            << *( b + offset1 ) << '\n';
29
30    // output array b using bPtr and array su
31    cout << "\nPointer subscript notation\n";
32
33    for ( int j = 0; j < 4; j++ )
34       cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';
35
36    cout << "\nPointer/offset notation\n";
37
38    // output array b using bPtr and pointer/offset notation
39    for ( int offset2 = 0; offset2 < 4; offset2++ )
40       cout << "*(bPtr + " << offset2 << ") = "
41            << *( bPtr + offset2 ) << '\n';
42
43    return 0;  // indicates successful termination
44
45 } // end main
```

Using array name and pointer/offset notation.

Using pointer subscript notation.

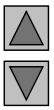
Using **`bPtr`** and pointer/offset notation.



Outline

fig05_20.cpp output (1 of 1)

```
Array b printed with:

Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

Pointer/offset notation where the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40
```

Outline

fig05_21.cpp (1 of 2)

```
1  // Fig. 5.21: fig05_21.cpp
2  // Copying a string using array notation
3  // and pointer notation.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  void copy1( char *, const char * );  // prototype
10 void copy2( char *, const char * );  // prototype
11
12 int main()
13 {
14    char string1[ 10 ];
15    char *string2 = "Hello";
16    char string3[ 10 ];
17    char string4[] = "Good Bye";
18
19    copy1( string1, string2 );
20    cout << "string1 = " << string1 << endl;
21
22    copy2( string3, string4 );
23    cout << "string3 = " << string3 << endl;
24
25    return 0;  // indicates successful termination
```

Outline

```
26
27 } // end main
28
29 // copy s2 to s1 using array notation
30 void copy1( char *s1, const char *s2 )
31 {
32    for ( int i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
33       ;   // do nothing in body
34
35 } // end function copy1
36
37 // copy s2 to s1 using pointer not
38 void copy2( char *s1, const char *
39 {
40    for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
41       ;   // do nothing in body
42
43 } // end function copy2
```

```
string1 = Hello
string3 = Good Bye
```

Use array subscript notation to copy string in **s2** to character array **s1**.

Use pointer notation to copy string in **s2** to character array in **s1**.

Increment both pointers to point to next elements in corresponding arrays.

05_21.cpp
of 2)

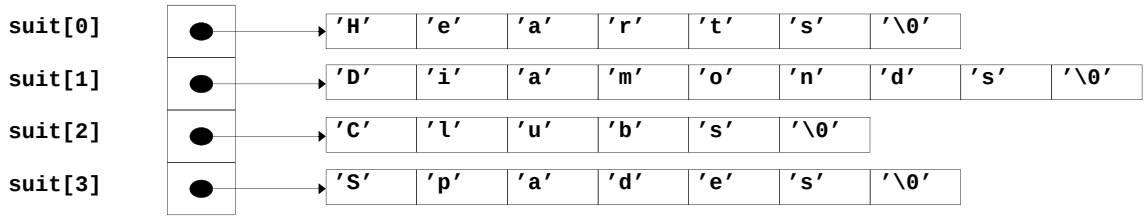
fig05_21.cpp
output (1 of 1)

Πίνακες δεικτών

- Οι πίνακες μπορεί να περιέχουν δείκτες
  - Π.χ. να αποθηκεύσουν έναν πίνακα με strings

```
char *suit[ 4 ] = {"Hearts", "Diamonds",
                   "Clubs", "Spades" };
```

  - Κάθε στοιχείο του **suit** δείχνει σε **`char *`** (a string)
  - Ο πίνακας δεν αποθηκεύει strings, μόνο δείκτες σε strings



  - Ο πίνακας **`suit`** έχει σταθερό μέγεθος, αλλά τα strings μπορεί να είναι οποιουδήποτε μεγέθους

Αναφορές

- Harvey M. Deitel, Paul J. Deitel, C++ How to Program, 4th Edition, Prentice Hall.
- Bjarne Stroustrup, The C++ Programming Language, Special Edition, Addison-Wesley.

Άσκηση-1

```
1  // ex05_21.cpp
2  #include <iostream>
3
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  void mystery1( char *, const char * );
9
10 int main()
11 {
12    char string1[ 80 ], string2[ 80 ];
13
14    cout << "Enter two strings: ";
15    cin >> string1 >> string2;
16    mystery1( string1, string2 );
17    cout << string1 << endl;
18
19    return 0;
20 }
21
22 void mystery1( char *s1, const char *s2 )
23 {
24    while ( *s1 != '\0' )
25       ++s1;
26
27    for ( ; *s1 = *s2; s1++, s2++ )
28       ;   // empty statement
29 }
```

- Τι κάνει το διπλανό πρόγραμμα?

Άσκηση-2

```
1  // ex05_22.cpp
2  #include <iostream>
3
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  int mystery2( const char * );
9
10 int main()
11 {
12    char string[ 80 ];
13
14    cout << "Enter a string: ";
15    cin >> string;
16    cout << mystery2( string ) << endl;
17
18    return 0;
19 }
20
21 int mystery2( const char *s )
22 {
23    int x;
24
25    for ( x = 0; *s != '\0'; s++ )
26       ++x;
27
28    return x;
29 }
```


- Τι κάνει το διπλανό πρόγραμμα?

Άσκηση-3

```
1  // ex05_30.cpp
2  #include <iostream>
3
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  bool mystery3( const char *, const char * );
9
10 int main()
11 {
12    char string1[ 80 ], string2[ 80 ];
13
14    cout << "Enter two strings: ";
15    cin >> string1 >> string2;
16    cout << "The result is "
17         << mystery3( string1, string2 ) << endl;
18
19    return 0;
20 }
21
22 bool mystery3( const char *s1, const char *s2 )
23 {
24    for ( ; *s1 != '\0' && *s2 != '\0'; s1++, s2++ )
25
26       if ( *s1 != *s2 )
27          return false;
28
29    return true;
30 }
```

- Τι κάνει το διπλανό πρόγραμμα?

Απάντηση-1

- Συνενώνει τα δύο string που δίνονται στην είσοδο και τα τυπώνει

Απάντηση-2

- Τυπώνει το μήκος του string

Απάντηση-3

- Ελέγχει την ισότητα των δύο string εισόδου