Αντικειμενοστραφείς Γλώσσες Προγραμματισμού C++ / ROOT

Ιωάννης Παπαδόπουλος

Τμήμα Φυσικής, Πανεπιστήμιο Ιωαννίνων

Νοέμβριος 2018

Περιεχόμενα

Περιεχόμενα

Γραμμή εντολής: παράμετροι και περιβάλλον

Γραμμή εντολής: παράμετροι και περιβάλλον

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

int main (
                int argc,

               char *argv[],

               char *env[]

         ) {

  for (int i=0;i<argc;i++) {
    cout << "Ορισμα " << i << ": "
         << argv[i] << endl;
  }

  int n=0;
  while (env[n]) {
    cout << "Μεταβλητή περιβάλλοντος " << n << ": "
         << env[n] << endl;
    n++;
  }

  return 0;
}

/*  Παράδειγμα εκτέλεσης του προγράμματος
Γραμμή εντολής:   ./main_args a "x y z" ελληνικά...
---> Εξοδος στην οθόνη:
Ορισμα 0: ./main_args
Ορισμα 1: a
Ορισμα 2: x y z
Ορισμα 3: ελληνικά...
Μεταβλητή περιβάλλοντος 0: LANG=en_US.UTF-8
Μεταβλητή περιβάλλοντος 1: LANGUAGE=en_US:en
*/
```

Η συνάρτηση **main** μπορεί να λάβει ορίσματα, που επιτρέπουν την ανάγνωση παραμέτρων από τη γραμμή εντολής κατά την εκτέλεση ενός προγράμματος.

- Δυνατές δηλώσεις της **main**:

```
int main();
```

```
int main( int argc, char* argv[] );
```

```
int main( int argc, char* argv[], char* env[] );
```

Γραμμή εντολής: παράμετροι και περιβάλλον

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

int main (
                int argc,

               char *argv[],

               char *env[]

         ) {

  for (int i=0;i<argc;i++) {
    cout << "Ορισμα " << i << ": "
         << argv[i] << endl;
  }

  int n=0;
  while (env[n]) {
    cout << "Μεταβλητή περιβάλλοντος " << n << ": "
         << env[n] << endl;
    n++;
  }

  return 0;
}

/*  Παράδειγμα εκτέλεσης του προγράμματος
Γραμμή εντολής:   ./main_args a "x y z" ελληνικά...
---> Εξοδος στην οθόνη:
Ορισμα 0: ./main_args
Ορισμα 1: a
Ορισμα 2: x y z
Ορισμα 3: ελληνικά...
Μεταβλητή περιβάλλοντος 0: LANG=en_US.UTF-8
Μεταβλητή περιβάλλοντος 1: LANGUAGE=en_US:en
*/
```

Η συνάρτηση **main** μπορεί να λάβει ορίσματα, που επιτρέπουν την ανάγνωση παραμέτρων από τη γραμμή εντολής κατά την εκτέλεση ενός προγράμματος.

γρ. 5 Ο ακέραιος **argc** δίνει τον αριθμό παραμέτρων που παρέχονται κατά την εκτέλεση στη γραμμή εντολής (συμπεριλαμβανομένου και του ονόματος του εκτελέσιμου αρχείου).

Γραμμή εντολής: παράμετροι και περιβάλλον

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

int main (
                int argc,

               char *argv[],

               char *env[]

         ) {

  for (int i=0;i<argc;i++) {
    cout << "Ορισμα " << i << ": "
         << argv[i] << endl;
  }

  int n=0;
  while (env[n]) {
    cout << "Μεταβλητή περιβάλλοντος " << n << ": "
         << env[n] << endl;
    n++;
  }

  return 0;
}

/*  Παράδειγμα εκτέλεσης του προγράμματος
Γραμμή εντολής:   ./main_args a "x y z" ελληνικά...
---> Εξοδος στην οθόνη:
Ορισμα 0: ./main_args
Ορισμα 1: a
Ορισμα 2: x y z
Ορισμα 3: ελληνικά...
Μεταβλητή περιβάλλοντος 0: LANG=en_US.UTF-8
Μεταβλητή περιβάλλοντος 1: LANGUAGE=en_US:en
*/
```

Η συνάρτηση **main** μπορεί να λάβει ορίσματα, που επιτρέπουν την ανάγνωση παραμέτρων από τη γραμμή εντολής κατά την εκτέλεση ενός προγράμματος.

γρ. 5 Ο ακέραιος **argc** δίνει τον αριθμό παραμέτρων που παρέχονται κατά την εκτέλεση στη γραμμή εντολής (συμπεριλαμβανομένου και του ονόματος του εκτελέσιμου αρχείου).

γρ. 7 Ο **argv** είναι ένας πίνακας C-strings (multi-byte, τερματιζόμενα με '\0'), πλήθους argc. Το argv[0] δίνει το όνομα του εκτελέσιμου αρχείου. Τα argv[1] έως argv[argc-1] δίνουν την αντίστοιχη παράμετρο της γραμμής εντολής.

Γραμμή εντολής: παράμετροι και περιβάλλον

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

int main (
                int argc,

               char *argv[],

               char *env[]

         ) {

  for (int i=0;i<argc;i++) {
    cout << "Ορισμα " << i << ": "
         << argv[i] << endl;
  }

  int n=0;
  while (env[n]) {
    cout << "Μεταβλητή περιβάλλοντος " << n << ": "
         << env[n] << endl;
    n++;
  }

  return 0;
}

/*  Παράδειγμα εκτέλεσης του προγράμματος
Γραμμή εντολής:   ./main_args a "x y z" ελληνικά...
---> Εξοδος στην οθόνη:
Ορισμα 0: ./main_args
Ορισμα 1: a
Ορισμα 2: x y z
Ορισμα 3: ελληνικά...
Μεταβλητή περιβάλλοντος 0: LANG=en_US.UTF-8
Μεταβλητή περιβάλλοντος 1: LANGUAGE=en_US:en
*/
```

Η συνάρτηση **main** μπορεί να λάβει ορίσματα, που επιτρέπουν την ανάγνωση παραμέτρων από τη γραμμή εντολής κατά την εκτέλεση ενός προγράμματος.

γρ. 5 Ο ακέραιος **argc** δίνει τον αριθμό παραμέτρων που παρέχονται κατά την εκτέλεση στη γραμμή εντολής (συμπεριλαμβανομένου και του ονόματος του εκτελέσιμου αρχείου).

γρ. 7 Ο **argv** είναι ένας πίνακας C-strings (multi-byte, τερματιζόμενα με '\0'), πλήθους argc. Το argv[0] δίνει το όνομα του εκτελέσιμου αρχείου. Τα argv[1] έως argv[argc-1] δίνουν την αντίστοιχη παράμετρο της γραμμής εντολής.

γρ. 9 Ο **env** είναι ένας πίνακας C-strings (multi-byte, τερματιζόμενα με '\0'), αγνώστου πλήθους, με τις όποιες μεταβλητές περιβάλλοντος εντός του οποίου εκτελείται το πρόγραμμα.

Δείκτες προς συναρτήσεις

Δείκτες προς συναρτήσεις

```cpp
#include <iostream>
using namespace std;
double sum(  double pf(double)  ,  int n  ) {
  double s=0;
  for (int i=1; i<=n; i++) s+=pf(i);
  return s;
}
double square(double x) { return x*x;   }
double   cube(double x) { return x*x*x; }

int main() {
  double (*pf1)(double)=square;
  double (*pf2)(double)=cube;
  cout << "   &pf1 : " <<            &pf1 <<endl;
  cout << "   *pf1 : " << (void*)    *pf1 <<endl;
  cout << "    pf1 : " << (void*)     pf1 <<endl;
  cout << "&square : " << (void*) &square <<endl;
  cout << "*square : " << (void*) *square <<endl;
  cout << " square : " << (void*)  square <<endl<<endl;

  cout << "   &pf2 : " <<            &pf2 <<endl;
  cout << "   *pf2 : " << (void*)    *pf2 <<endl;
  cout << "    pf2 : " << (void*)     pf2 <<endl;
  cout << "  &cube : " << (void*)   &cube <<endl;
  cout << "  *cube : " << (void*)   *cube <<endl;
  cout << "   cube : " << (void*)    cube <<endl<<endl;

  cout << "όροι / 1²+2²+... / 1³+2³+..." << endl;
  for (int i=0;i<=10;i++) {
    cout << i  << " " << sum(pf1,i)
               << " " << sum(pf2,i)
               << endl;
  }
  return 0;
}
```

Μία συνάρτηση (έστω η **func**) όταν ορίζεται μετατρέπεται αυτομάτως σε δείκτη, που δείχνει στη θέση μνήμης όπου βρίσκεται ο κώδικάς της. Έτσι το όνομά της ταυτίζεται με τη διεύθυνσή της αλλά και με το περιεχόμενο της διεύθυνσής της, δηλαδή: func ≡ &func ≡ *func

Δείκτες προς συναρτήσεις

```cpp
#include <iostream>
using namespace std;
double sum(  double pf(double)  ,  int n  ) {
  double s=0;
  for (int i=1; i<=n; i++) s+=pf(i);
  return s;
}
double square(double x) { return x*x;   }
double   cube(double x) { return x*x*x; }

int main() {
  double (*pf1)(double)=square;
  double (*pf2)(double)=cube;
  cout << "   &pf1 : " <<            &pf1 <<endl;
  cout << "   *pf1 : " << (void*)    *pf1 <<endl;
  cout << "    pf1 : " << (void*)     pf1 <<endl;
  cout << "&square : " << (void*) &square <<endl;
  cout << "*square : " << (void*) *square <<endl;
  cout << " square : " << (void*)  square <<endl<<endl;

  cout << "   &pf2 : " <<            &pf2 <<endl;
  cout << "   *pf2 : " << (void*)    *pf2 <<endl;
  cout << "    pf2 : " << (void*)     pf2 <<endl;
  cout << "  &cube : " << (void*)   &cube <<endl;
  cout << "  *cube : " << (void*)   *cube <<endl;
  cout << "   cube : " << (void*)    cube <<endl<<endl;

  cout << "όροι / 1²+2²+... / 1³+2³+..." << endl;
  for (int i=0;i<=10;i++) {
    cout << i  << " " << sum(pf1,i)
               << " " << sum(pf2,i)
               << endl;
  }
  return 0;
}
```

Μία συνάρτηση (έστω η **func**) όταν ορίζεται μετατρέπεται αυτομάτως σε δείκτη, που δείχνει στη θέση μνήμης όπου βρίσκεται ο κώδικάς της. Έτσι το όνομά της ταυτίζεται με τη διεύθυνσή της αλλά και με το περιεχόμενο της διεύθυνσής της, δηλαδή: func ≡ &func ≡ *func

Μπορούμε να ορίσουμε δείκτες προς συναρτήσεις, και να τους δώσουμε ως τιμή μία συνάρτηση. Τότε ο δείκτης (έστω ο pf) θα δείχνει στη συνάρτηση και το ίδιο θα συμβαίνει και με την αποαναφοροποιημένη του τιμή, δηλαδή: pf ≡ *pf

Δείκτες προς συναρτήσεις

```
#include <iostream>
using namespace std;
double sum(  double pf(double)  ,  int n  ) {
  double s=0;
  for (int i=1; i<=n; i++) s+=pf(i);
  return s;
}
double square(double x) { return x*x;   }
double   cube(double x) { return x*x*x; }

int main() {
  double (*pf1)(double)=square;
  double (*pf2)(double)=cube;
  cout << "   &pf1 : " <<            &pf1 <<endl;
  cout << "   *pf1 : " << (void*)    *pf1 <<endl;
  cout << "    pf1 : " << (void*)     pf1 <<endl;
  cout << "&square : " << (void*) &square <<endl;
  cout << "*square : " << (void*) *square <<endl;
  cout << " square : " << (void*)  square <<endl<<endl;

  cout << "   &pf2 : " <<            &pf2 <<endl;
  cout << "   *pf2 : " << (void*)    *pf2 <<endl;
  cout << "    pf2 : " << (void*)     pf2 <<endl;
  cout << "  &cube : " << (void*)   &cube <<endl;
  cout << "  *cube : " << (void*)   *cube <<endl;
  cout << "   cube : " << (void*)    cube <<endl<<endl;

  cout << "όροι / 1²+2²+... / 1³+2³+..." << endl;
  for (int i=0;i<=10;i++) {
    cout << i  << " " << sum(pf1,i)
               << " " << sum(pf2,i)
               << endl;
  }
  return 0;
}
```

Μία συνάρτηση (έστω η **func**) όταν ορίζεται μετατρέπεται αυτομάτως σε δείκτη, που δείχνει στη θέση μνήμης όπου βρίσκεται ο κώδικάς της. Έτσι το όνομά της ταυτίζεται με τη διεύθυνσή της αλλά και με το περιεχόμενο της διεύθυνσής της, δηλαδή: func ≡ &func ≡ *func

Μπορούμε να ορίσουμε δείκτες προς συναρτήσεις, και να τους δώσουμε ως τιμή μία συνάρτηση. Τότε ο δείκτης (έστω ο pf) θα δείχνει στη συνάρτηση και το ίδιο θα συμβαίνει και με την αποαναφοροποιημένη του τιμή, δηλαδή: pf ≡ *pf

Στο διπλανό παράδειγμα, τυπώνονται οι διάφορες διευθύνσεις για την κατάδειξη των παραπάνω.

Για ένα πιο σύνθετο παράδειγμα δείτε εδώ.

Δείκτες προς συναρτήσεις

```
/*
   &pf1 : 0x7ffceb71a648
   *pf1 : 0x563ab3796a5b
    pf1 : 0x563ab3796a5b
&square : 0x563ab3796a5b
*square : 0x563ab3796a5b
 square : 0x563ab3796a5b

   &pf2 : 0x7ffceb71a650
   *pf2 : 0x563ab3796a70
    pf2 : 0x563ab3796a70
  &cube : 0x563ab3796a70
  *cube : 0x563ab3796a70
   cube : 0x563ab3796a70

όροι / 1²+2²+... / 1³+2³+...
0 0 0
1 1 1
2 5 9
3 14 36
4 30 100
5 55 225
6 91 441
7 140 784
8 204 1296
9 285 2025
10 385 3025
*/
```

Αποτελέσματα του προηγούμενου παραδείγματος (οι θέσεις μνήμης είναι ενδεικτικές, και αλλάζουν κάθε φορά που εκτελείται το πρόγραμμα).

Δείκτες προς συναρτήσεις

```
1  /*
2     &pf1 : 0x7ffceb71a648
3     *pf1 : 0x563ab3796a5b
4      pf1 : 0x563ab3796a5b
5  &square : 0x563ab3796a5b
6  *square : 0x563ab3796a5b
7   square : 0x563ab3796a5b
8
9     &pf2 : 0x7ffceb71a650
10    *pf2 : 0x563ab3796a70
11     pf2 : 0x563ab3796a70
12   &cube : 0x563ab3796a70
13   *cube : 0x563ab3796a70
14    cube : 0x563ab3796a70
15
16 όροι / 1²+2²+... / 1³+2³+...
17 0 0 0
18 1 1 1
19 2 5 9
20 3 14 36
21 4 30 100
22 5 55 225
23 6 91 441
24 7 140 784
25 8 204 1296
26 9 285 2025
27 10 385 3025
28 */
```

Αποτελέσματα του προηγούμενου παραδείγματος (οι θέσεις μνήμης είναι ενδεικτικές, και αλλάζουν κάθε φορά που εκτελείται το πρόγραμμα).

Οι γραμμές 3-7 δείχνουν την ισοδυναμία των διαφόρων αναπαραστάσεων που τελικά δείχνουν στην ίδια θέση μνήμης, όπου βρίσκεται ο κώδικας της συνάρτησης square.

Δείκτες προς συναρτήσεις

```
1  /*
2     &pf1 : 0x7ffceb71a648
3     *pf1 : 0x563ab3796a5b
4      pf1 : 0x563ab3796a5b
5  &square : 0x563ab3796a5b
6  *square : 0x563ab3796a5b
7   square : 0x563ab3796a5b
8
9     &pf2 : 0x7ffceb71a650
10    *pf2 : 0x563ab3796a70
11     pf2 : 0x563ab3796a70
12   &cube : 0x563ab3796a70
13   *cube : 0x563ab3796a70
14    cube : 0x563ab3796a70
15
16 όροι / 1²+2²+... / 1³+2³+...
17 0 0 0
18 1 1 1
19 2 5 9
20 3 14 36
21 4 30 100
22 5 55 225
23 6 91 441
24 7 140 784
25 8 204 1296
26 9 285 2025
27 10 385 3025
28 */
```

Αποτελέσματα του προηγούμενου παραδείγματος (οι θέσεις μνήμης είναι ενδεικτικές, και αλλάζουν κάθε φορά που εκτελείται το πρόγραμμα).

Οι γραμμές 3-7 δείχνουν την ισοδυναμία των διαφόρων αναπαραστάσεων που τελικά δείχνουν στην ίδια θέση μνήμης, όπου βρίσκεται ο κώδικας της συνάρτησης square.

Ομοίως, οι γραμμές 10-14 δείχνουν την ισοδυναμία των διαφόρων αναπαραστάσεων που τελικά δείχνουν στην ίδια θέση μνήμης, όπου βρίσκεται ο κώδικας της συνάρτησης cube.

Δείκτες προς συναρτήσεις

```
1  /*
2     &pf1 : 0x7ffceb71a648
3     *pf1 : 0x563ab3796a5b
4      pf1 : 0x563ab3796a5b
5  &square : 0x563ab3796a5b
6  *square : 0x563ab3796a5b
7   square : 0x563ab3796a5b
8
9     &pf2 : 0x7ffceb71a650
10    *pf2 : 0x563ab3796a70
11     pf2 : 0x563ab3796a70
12   &cube : 0x563ab3796a70
13   *cube : 0x563ab3796a70
14    cube : 0x563ab3796a70
15
16 όροι / 1²+2²+... / 1³+2³+...
17 0 0 0
18 1 1 1
19 2 5 9
20 3 14 36
21 4 30 100
22 5 55 225
23 6 91 441
24 7 140 784
25 8 204 1296
26 9 285 2025
27 10 385 3025
28 */
```

Αποτελέσματα του προηγούμενου παραδείγματος (οι θέσεις μνήμης είναι ενδεικτικές, και αλλάζουν κάθε φορά που εκτελείται το πρόγραμμα).

Οι γραμμές 3-7 δείχνουν την ισοδυναμία των διαφόρων αναπαραστάσεων που τελικά δείχνουν στην ίδια θέση μνήμης, όπου βρίσκεται ο κώδικας της συνάρτησης square.

Ομοίως, οι γραμμές 10-14 δείχνουν την ισοδυναμία των διαφόρων αναπαραστάσεων που τελικά δείχνουν στην ίδια θέση μνήμης, όπου βρίσκεται ο κώδικας της συνάρτησης cube.

Ακολουθούν (γραμμές 16-28) τα αποτελέσματα εκτέλεσης του βρόχου for εντός του οποίου καλείται η συνάρτηση sum, μία φορά με τη συνάρτηση square και μία με τη συνάρτηση cube.

Αναφορές (References)

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.
- Οι αναφορές **πρέπει να αρχικοποιούνται**.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.
- Οι αναφορές **πρέπει να αρχικοποιούνται**.
- Όσο υφίσταται, μία αναφορά **συνδέεται με την ίδια πάντα μεταβλητή**.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.
- Οι αναφορές **πρέπει να αρχικοποιούνται**.
- Όσο υφίσταται, μία αναφορά **συνδέεται με την ίδια πάντα μεταβλητή**.
- Μοιάζουν με τους δείκτες, όμως δεν απαιτούν αποαναφοροποίηση όπως αυτοί.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.
- Οι αναφορές **πρέπει να αρχικοποιούνται**.
- Όσο υφίσταται, μία αναφορά **συνδέεται με την ίδια πάντα μεταβλητή**.
- Μοιάζουν με τους δείκτες, όμως δεν απαιτούν αποαναφοροποίηση όπως αυτοί.

```
int   i = 100 ;  // το i έχει τιμή 100
int &ri = i ;
     ri = 3 ;    // το i έχει τιμή 3
```

- Οι αναφορές l-value **δηλώνονται μέσω του τελεστή** `&` πριν το αναγνωριστικό.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.
- Οι αναφορές **πρέπει να αρχικοποιούνται**.
- Όσο υφίσταται, μία αναφορά **συνδέεται με την ίδια πάντα μεταβλητή**.
- Μοιάζουν με τους δείκτες, όμως δεν απαιτούν αποαναφοροποίηση όπως αυτοί.

```
int   i = 100 ;  // το i έχει τιμή 100
int &ri = i ;
    ri = 3 ;    // το i έχει τιμή 3
```

```
int   i = 100 ;
int &ri ;             // σφάλμα
```

- Οι αναφορές l-value **δηλώνονται μέσω του τελεστή** `&` πριν το αναγνωριστικό.
- *Σφάλμα:* Δεν έγινε αρχικοποίηση της αναφοράς.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.
- Οι αναφορές **πρέπει να αρχικοποιούνται**.
- Όσο υφίσταται, μία αναφορά **συνδέεται με την ίδια πάντα μεταβλητή**.
- Μοιάζουν με τους δείκτες, όμως δεν απαιτούν αποαναφοροποίηση όπως αυτοί.

```
int   i = 100 ;  // το i έχει τιμή 100
int &ri = i ;
    ri = 3 ;    // το i έχει τιμή 3
```

- Οι αναφορές l-value **δηλώνονται μέσω του τελεστή** `&` πριν το αναγνωριστικό.

```
int   i = 100 ;
int &ri ;              // σφάλμα
```

- *Σφάλμα:* Δεν έγινε αρχικοποίηση της αναφοράς.

```
double   d = 2.5 ;
int    &ri = d ;       // σφάλμα
```

- *Σφάλμα:* Ασυμφωνία τύπων.

Αναφορές l-value (l-value references)

- Οι αναφορές l-value είναι **συνώνυμα (aliases) μεταβλητών**, δηλαδή μπορούν να χρησιμοποιηθούν για πρόσβαση με άλλο όνομα στη μεταβλητή.
- Μεταβολή της τιμής μίας αναφοράς σημαίνει μεταβολή της τιμής της αρχικής μεταβλητής στην οποία αναφέρεται.
- Οι αναφορές **πρέπει να αρχικοποιούνται**.
- Όσο υφίσταται, μία αναφορά **συνδέεται με την ίδια πάντα μεταβλητή**.
- Μοιάζουν με τους δείκτες, όμως δεν απαιτούν αποαναφοροποίηση όπως αυτοί.

```
int   i = 100 ;  // το i έχει τιμή 100
int &ri = i ;
    ri = 3 ;    // το i έχει τιμή 3
```

```
int   i = 100 ;
int &ri ;              // σφάλμα
```

```
double   d = 2.5 ;
int    &ri = d ;       // σφάλμα
```

```
int &ri = 33 ;         // σφάλμα
```

- Οι αναφορές l-value **δηλώνονται μέσω του τελεστή** `&` πριν το αναγνωριστικό.
- *Σφάλμα:* Δεν έγινε αρχικοποίηση της αναφοράς.
- *Σφάλμα:* Ασυμφωνία τύπων.
- *Σφάλμα:* Μία αναφορά l-value δεν μπορεί να συνδεθεί με μία σταθερά.

Αναφορές l-value ως ορίσματα συναρτήσεων

Έχουμε ήδη δει ότι:

- Όταν τα ορίσματα μίας συνάρτησης περνούν μέσω τιμής (pass by value), οι χειρισμοί τους εντός του σώματος της συνάρτησης δεν μεταβάλλουν τις τιμές των αρχικών μεταβλητών, οι οποίες εμφανίζονται στην εντολή κλήσης της συνάρτησης.

Αναφορές l-value ως ορίσματα συναρτήσεων

Έχουμε ήδη δει ότι:

- Όταν τα ορίσματα μίας συνάρτησης περνούν μέσω τιμής (pass by value), οι χειρισμοί τους εντός του σώματος της συνάρτησης δεν μεταβάλλουν τις τιμές των αρχικών μεταβλητών, οι οποίες εμφανίζονται στην εντολή κλήσης της συνάρτησης.
- Είναι δυνατόν να μεταβληθούν οι τιμές των αρχικών μεταβλητών, αν χρησιμοποιηθούν δείκτες για το πέρασμα των ορισμάτων στην συνάρτηση (pass by pointer).

Αναφορές l-value ως ορίσματα συναρτήσεων

Έχουμε ήδη δει ότι:

- Όταν τα ορίσματα μίας συνάρτησης περνούν μέσω τιμής (pass by value), οι χειρισμοί τους εντός του σώματος της συνάρτησης δεν μεταβάλλουν τις τιμές των αρχικών μεταβλητών, οι οποίες εμφανίζονται στην εντολή κλήσης της συνάρτησης.
- Είναι δυνατόν να μεταβληθούν οι τιμές των αρχικών μεταβλητών, αν χρησιμοποιηθούν δείκτες για το πέρασμα των ορισμάτων στην συνάρτηση (pass by pointer).
- Με τη χρησιμοποίηση δεικτών, αποφεύγεται η αντιγραφή μεγάλου όγκου δεδομένων κατά την κλήση συναρτήσεων, επιταχύνοντας την εκτέλεση του προγράμματος.

Αναφορές l-value ως ορίσματα συναρτήσεων

Έχουμε ήδη δει ότι:

- Όταν τα ορίσματα μίας συνάρτησης περνούν μέσω τιμής (pass by value), οι χειρισμοί τους εντός του σώματος της συνάρτησης δεν μεταβάλλουν τις τιμές των αρχικών μεταβλητών, οι οποίες εμφανίζονται στην εντολή κλήσης της συνάρτησης.
- Είναι δυνατόν να μεταβληθούν οι τιμές των αρχικών μεταβλητών, αν χρησιμοποιηθούν δείκτες για το πέρασμα των ορισμάτων στην συνάρτηση (pass by pointer).
- Με τη χρησιμοποίηση δεικτών, αποφεύγεται η αντιγραφή μεγάλου όγκου δεδομένων κατά την κλήση συναρτήσεων, επιταχύνοντας την εκτέλεση του προγράμματος.

Μπορούμε να χρησιμοποιήσουμε αναφορές:

- Η χρήση αναφορών για το πέρασμα ορισμάτων στις συναρτήσεις (pass by reference) παρέχει τα πλεονεκτήματα των δεικτών, χωρίς όμως την πολύπλοκη σύνταξη που απαιτεί η αποαναφοροποίησή τους.

Αναφορές l-value ως ορίσματα συναρτήσεων

Έχουμε ήδη δει ότι:

- Όταν τα ορίσματα μίας συνάρτησης περνούν μέσω τιμής (pass by value), οι χειρισμοί τους εντός του σώματος της συνάρτησης δεν μεταβάλλουν τις τιμές των αρχικών μεταβλητών, οι οποίες εμφανίζονται στην εντολή κλήσης της συνάρτησης.
- Είναι δυνατόν να μεταβληθούν οι τιμές των αρχικών μεταβλητών, αν χρησιμοποιηθούν δείκτες για το πέρασμα των ορισμάτων στην συνάρτηση (pass by pointer).
- Με τη χρησιμοποίηση δεικτών, αποφεύγεται η αντιγραφή μεγάλου όγκου δεδομένων κατά την κλήση συναρτήσεων, επιταχύνοντας την εκτέλεση του προγράμματος.

Μπορούμε να χρησιμοποιήσουμε αναφορές:

- Η χρήση αναφορών για το πέρασμα ορισμάτων στις συναρτήσεις (pass by reference) παρέχει τα πλεονεκτήματα των δεικτών, χωρίς όμως την πολύπλοκη σύνταξη που απαιτεί η αποαναφοροποίησή τους.
- Μπορεί να απαγορευτεί η μεταβολή της αρχικής μεταβλητής ορίζοντας την αναφορά ως `const type &`.

Αναφορές (r-value references)

- Από την τυποποίηση C++11 της C++ το 2011 και μετέπειτα, επιτρέπεται η χρήση αναφορών r-value.

Αναφορές (r-value references)

- Από την τυποποίηση C++11 της C++ το 2011 και μετέπειτα, επιτρέπεται η χρήση αναφορών r-value.
- Οι αναφορές r-value, διαφέρουν ως προς τις αναφορές l-value που έχουμε δει έως τώρα, στο ότι μπορούν να αρχικοποιηθούν σε τιμές άλλες από μεταβλητές.

Αναφορές (r-value references)

- Από την τυποποίηση C++11 της C++ το 2011 και μετέπειτα, επιτρέπεται η χρήση αναφορών r-value.
- Οι αναφορές r-value, διαφέρουν ως προς τις αναφορές l-value που έχουμε δει έως τώρα, στο ότι μπορούν να αρχικοποιηθούν σε τιμές άλλες από μεταβλητές.
- Οι αναφορές r-value **δηλώνονται μέσω του τελεστή** `&&` πριν το αναγνωριστικό.

Αναφορές (r-value references)

- Από την τυποποίηση C++11 της C++ το 2011 και μετέπειτα, επιτρέπεται η χρήση αναφορών r-value.
- Οι αναφορές r-value, διαφέρουν ως προς τις αναφορές l-value που έχουμε δει έως τώρα, στο ότι μπορούν να αρχικοποιηθούν σε τιμές άλλες από μεταβλητές.
- Οι αναφορές r-value **δηλώνονται μέσω του τελεστή** `&&` πριν το αναγνωριστικό.

```
int  &Lri = 33 ;          // σφάλμα
```

- *Σφάλμα:* Μία αναφορά l-value δεν μπορεί να συνδεθεί με μία σταθερά.

Αναφορές (r-value references)

- Από την τυποποίηση C++11 της C++ το 2011 και μετέπειτα, επιτρέπεται η χρήση αναφορών r-value.
- Οι αναφορές r-value, διαφέρουν ως προς τις αναφορές l-value που έχουμε δει έως τώρα, στο ότι μπορούν να αρχικοποιηθούν σε τιμές άλλες από μεταβλητές.
- Οι αναφορές r-value **δηλώνονται μέσω του τελεστή** && πριν το αναγνωριστικό.

```
int  &Lri = 33 ;          // σφάλμα
```

- *Σφάλμα:* Μία αναφορά l-value δεν μπορεί να συνδεθεί με μία σταθερά.

```
int &&Rri = 33 ;          // OK
```

- *OK:* Αρχικοποίηση σε τιμή rvalue.

Αναφορές (r-value references)

- Από την τυποποίηση C++11 της C++ το 2011 και μετέπειτα, επιτρέπεται η χρήση αναφορών r-value.
- Οι αναφορές r-value, διαφέρουν ως προς τις αναφορές l-value που έχουμε δει έως τώρα, στο ότι μπορούν να αρχικοποιηθούν σε τιμές άλλες από μεταβλητές.
- Οι αναφορές r-value **δηλώνονται μέσω του τελεστή** && πριν το αναγνωριστικό.

```
int  &Lri = 33 ;          // σφάλμα
```

- *Σφάλμα:* Μία αναφορά l-value δεν μπορεί να συνδεθεί με μία σταθερά.

```
int &&Rri = 33 ;          // OK
```

- *OK:* Αρχικοποίηση σε τιμή rvalue.

```
int      i = 2 ;
int &&Rri = i ;           // σφάλμα
```

- *Σφάλμα:* Απαγορεύεται η αρχικοποίηση σε τιμή lvalue.

Αναφορές (r-value references)

- Από την τυποποίηση C++11 της C++ το 2011 και μετέπειτα, επιτρέπεται η χρήση αναφορών r-value.
- Οι αναφορές r-value, διαφέρουν ως προς τις αναφορές l-value που έχουμε δει έως τώρα, στο ότι μπορούν να αρχικοποιηθούν σε τιμές άλλες από μεταβλητές.
- Οι αναφορές r-value **δηλώνονται μέσω του τελεστή** && πριν το αναγνωριστικό.

```
int  &Lri = 33 ;          // σφάλμα
```

- *Σφάλμα:* Μία αναφορά l-value δεν μπορεί να συνδεθεί με μία σταθερά.

```
int &&Rri = 33 ;          // OK
```

- *OK:* Αρχικοποίηση σε τιμή rvalue.

```
int      i = 2 ;
int &&Rri = i ;           // σφάλμα
```

- *Σφάλμα:* Απαγορεύεται η αρχικοποίηση σε τιμή lvalue.

```
int      i = 2 ;
int &&Rri = i+0 ;         // OK
```

- *OK:* Η τιμή **i+0** είναι τιμή rvalue.

Αναφορές l-value : Άσκηση

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

void swap ( int* , int* );

int main() {

  int i = 111;
  int j = 222;

  int *pi = &i;
  int *pj = &j;

  cout <<   "i = " << i
       << ", j = " << j << endl;

  swap( pi, pj );

  cout <<   "i = " << i
       << ", j = " << j << endl;

  return 0;

}

void swap( int *a , int *b ) {
  int k = *a;
  *a = *b;
  *b = k;
  return;
}
```

Έστω το διπλανό πρόγραμμα που υλοποιεί την συνάρτηση swap χρησιμοποιώντας δείκτες για να εναλλάξει τις τιμές δύο ακεραίων μεταβλητών.

Να **μετατρέψετε το πρόγραμμα** έτσι, ώστε να έχει την ίδια λειτουργικότητα, αλλά **αντί για δείκτες να χρησιμοποιηθούν αναφορές l-value** (δηλαδή αντί του περάσματος ορισμάτων μέσω δεικτών, να γίνει με πέρασμα ορισμάτων μέσω αναφοράς).

Διαχείριση μνήμης

Μνήμη stack και heap

- Η μνήμη που χρησιμοποιεί ένα πρόγραμμα, πέραν από τον χώρο που καταλαμβάνει ο εκτελέσιμος κώδικας μηχανής και δεδομένα (π.χ. σταθερές, C-strings, κ.ο.κ.), διακρίνεται
  - στη μνήμη **stack** (στοίβας) και
  - στη μνημη **heap** (σωρού).

Μνήμη stack και heap

- Η μνήμη που χρησιμοποιεί ένα πρόγραμμα, πέραν από τον χώρο που καταλαμβάνει ο εκτελέσιμος κώδικας μηχανής και δεδομένα (π.χ. σταθερές, C-strings, κ.ο.κ.), διακρίνεται
  - στη μνήμη **stack** (στοίβας) και
  - στη μνημη **heap** (σωρού).
- Μνήμη **stack**
  - Χρησιμοποιείται για την **αποθήκευση τοπικών μεταβλητών**.
  - Έχει δομή LIFO (Last-In, First-Out).
  - Είναι εύκολα διαχειρίσιμη εντός του προγράμματος και **πολύ γρήγορη** (χωρίς κλήσεις προς το λειτουργικό σύστημα).
  - Έχει **μικρό μέγεθος** (από μερικά kByte, ως κάποια MByte), που καθορίζεται από το λειτουργικό σύστημα όταν το πρόγραμμα ξεκινά να εκτελείται.

Μνήμη stack και heap

- Η μνήμη που χρησιμοποιεί ένα πρόγραμμα, πέραν από τον χώρο που καταλαμβάνει ο εκτελέσιμος κώδικας μηχανής και δεδομένα (π.χ. σταθερές, C-strings, κ.ο.κ.), διακρίνεται
  - στη μνήμη **stack** (στοίβας) και
  - στη μνημη **heap** (σωρού).
- Μνήμη **stack**
  - Χρησιμοποιείται για την **αποθήκευση τοπικών μεταβλητών**.
  - Έχει δομή LIFO (Last-In, First-Out).
  - Είναι εύκολα διαχειρίσιμη εντός του προγράμματος και **πολύ γρήγορη** (χωρίς κλήσεις προς το λειτουργικό σύστημα).
  - Έχει **μικρό μέγεθος** (από μερικά kByte, ως κάποια MByte), που καθορίζεται από το λειτουργικό σύστημα όταν το πρόγραμμα ξεκινά να εκτελείται.
- Μνήμη **heap**
  - Χρησιμοποιείται για την **δυναμική δέσμευση μνήμης**.
  - Την διαχειρίζεται το λειτουργικό σύστημα, κατά συνέπεια είναι πιο αργή λόγω των κλήσεων προς το λειτουργικό σύστημα.
  - Έχει γενικά **μεγάλο μέγεθος** (όση και η ελεύθερη μνήμη του υπολογιστή), και είναι διαθέσιμη προς δέσμευση και χρησιμοποίηση.

Τελεστές new και delete

- Δέσμευση μνήμης stack κατά τη μεταγλώττιση:
  παραμένει δεσμευμένη καθόλη τη διάρκεια της εκτέλεσης του προγράμματος.

```
double d = 33.5 ;     // δέσμευση μνήμης stack και αρχικοποίηση
double *pd = &d ;     // δήλωση δείκτη και αρχικοποίηση
```

Τελεστές new και delete

- Δέσμευση μνήμης stack κατά τη μεταγλώττιση:
  παραμένει δεσμευμένη καθόλη τη διάρκεια της εκτέλεσης του προγράμματος.

```
double d = 33.5 ;    // δέσμευση μνήμης stack και αρχικοποίηση
double *pd = &d ;    // δήλωση δείκτη και αρχικοποίηση
```

- Δυναμική δέσμευση μνήμης heap (κατά την εκτέλεση) μέσω του τελεστή **new**:

```
double *pd ;         // δήλωση δείκτη
pd = new double;     // δέσμευση μνήμης heap
*pd = 33.5 ;         // απόδοση τιμής
```

```
double *pd = new double(33.5);     // ή ισοδύναμα (all-in-one...)
```

Τελεστές new και delete

- Δέσμευση μνήμης stack κατά τη μεταγλώττιση:
  παραμένει δεσμευμένη καθόλη τη διάρκεια της εκτέλεσης του προγράμματος.

```
double d = 33.5 ;    // δέσμευση μνήμης stack και αρχικοποίηση
double *pd = &d ;    // δήλωση δείκτη και αρχικοποίηση
```

- Δυναμική δέσμευση μνήμης heap (κατά την εκτέλεση) μέσω του τελεστή **new**:

```
double *pd ;         // δήλωση δείκτη
pd = new double;     // δέσμευση μνήμης heap
*pd = 33.5 ;         // απόδοση τιμής
```

```
double *pd = new double(33.5);     // ή ισοδύναμα (all-in-one...)
```

- Αποδέσμευση μνήμης heap μέσω του τελεστή **delete**:

```
double *pd = new double(33.5);     // δέσμευση μνήμης heap
cout << *pd << endl;
delete pd ;                        // αποδέσμευση μνήμης heap
```

- Το delete μπορεί να χρησιμοποιηθεί εφόσον έχει προηγηθεί κάπου στο τρέχον μπλοκ σχετική εντολή **new**.

Τελεστές new[] και delete[]

- Δέσμευση μνήμης stack για πίνακα κατά τη μεταγλώττιση:
  παραμένει δεσμευμένη καθόλη τη διάρκεια της εκτέλεσης του προγράμματος,
  ακόμη κι αν δεν χρησιμοποιηθεί σε όλη την έκταση του προγράμματος.

```
double a[1000] ;    // δέσμευση μνήμης stack
```

Τελεστές new[] και delete[]

- Δέσμευση μνήμης stack για πίνακα κατά τη μεταγλώττιση:
  παραμένει δεσμευμένη καθόλη τη διάρκεια της εκτέλεσης του προγράμματος,
  ακόμη κι αν δεν χρησιμοποιηθεί σε όλη την έκταση του προγράμματος.

```
double a[1000] ;    // δέσμευση μνήμης stack
```

- Δυναμική δέσμευση μνήμης heap για πίνακα (κατά την εκτέλεση, τελεστής **new[]**):

```
int n;
cout << "Δώσε τον αριθμό στοιχείων του πίνακα: "
cin >> n ;
double *a = new double[n] ; // δήλωση δείκτη πίνακα, δέσμευση μνήμης heap
```

Τελεστές new[] και delete[]

- Δέσμευση μνήμης stack για πίνακα κατά τη μεταγλώττιση:
  παραμένει δεσμευμένη καθόλη τη διάρκεια της εκτέλεσης του προγράμματος,
  ακόμη κι αν δεν χρησιμοποιηθεί σε όλη την έκταση του προγράμματος.

```
double a[1000] ;    // δέσμευση μνήμης stack
```

- Δυναμική δέσμευση μνήμης heap για πίνακα (κατά την εκτέλεση, τελεστής **new[]**):

```
int n;
cout << "Δώσε τον αριθμό στοιχείων του πίνακα: "
cin >> n ;
double *a = new double[n] ; // δήλωση δείκτη πίνακα, δέσμευση μνήμης heap
```

- Αποδέσμευση μνήμης heap μέσω του τελεστή **delete**:

```
delete a[] ;                        // αποδέσμευση μνήμης heap
```

- Η εντολή **delete** μπορεί να χρησιμοποιηθεί, εφόσον έχει προηγηθεί,
  κάπου στο τρέχον μπλοκ, μία σχετική εντολή **new**.
- Η μνήμη heap που αποδεσμεύεται, αποδίδεται πίσω στο λειτουργικό σύστημα,
  και μπορεί να χρησιμοποιηθεί, πλέον, για κάποιον άλλο σκοπό.

C++ strings

C++ strings

- Η γενίκευση των C-strings (αλφαριθμητικών ακολουθιών) οδηγεί στην κλάση της C++ που ονομάζεται `string`.

C++ strings

- Η γενίκευση των C-strings (αλφαριθμητικών ακολουθιών) οδηγεί στην κλάση της C++ που ονομάζεται `string`.
- Τα C++ strings μπορούν να συγκριθούν, να επεκταθούν, να συγκολληθούν, να χρησιμοποιηθούν σε αναζητήσεις χαρακτήρων και επιμέρους strings.

C++ strings

- Η γενίκευση των C-strings (αλφαριθμητικών ακολουθιών) οδηγεί στην κλάση της C++ που ονομάζεται `string`.
- Τα C++ strings μπορούν να συγκριθούν, να επεκταθούν, να συγκολληθούν, να χρησιμοποιηθούν σε αναζητήσεις χαρακτήρων και επιμέρους strings.
- Οι επιμέρους χαρακτήρες ενός string είναι προσβάσιμοι μέσω του τελεστή [].

C++ strings

- Η γενίκευση των C-strings (αλφαριθμητικών ακολουθιών) οδηγεί στην κλάση της C++ που ονομάζεται `string`.
- Τα C++ strings μπορούν να συγκριθούν, να επεκταθούν, να συγκολληθούν, να χρησιμοποιηθούν σε αναζητήσεις χαρακτήρων και επιμέρους strings.
- Οι επιμέρους χαρακτήρες ενός string είναι προσβάσιμοι μέσω του τελεστή [].
- Μπορούν να χρησιμοποιηθούν με τους (υπερφορτωμένους) λογικούς τελεστές:
  $<$ $<=$ $>$ $>=$ $==$ $!=$

C++ strings

- Η γενίκευση των C-strings (αλφαριθμητικών ακολουθιών) οδηγεί στην κλάση της C++ που ονομάζεται `string`.
- Τα C++ strings μπορούν να συγκριθούν, να επεκταθούν, να συγκολληθούν, να χρησιμοποιηθούν σε αναζητήσεις χαρακτήρων και επιμέρους strings.
- Οι επιμέρους χαρακτήρες ενός string είναι προσβάσιμοι μέσω του τελεστή [].
- Μπορούν να χρησιμοποιηθούν με τους (υπερφορτωμένους) λογικούς τελεστές: $<$ $<=$ $>$ $>=$ $==$ $!=$
- Ένα string μπορεί να αρχικοποιηθεί από έναν πίνακα char.

C++ strings

- Η γενίκευση των C-strings (αλφαριθμητικών ακολουθιών) οδηγεί στην κλάση της C++ που ονομάζεται `string`.
- Τα C++ strings μπορούν να συγκριθούν, να επεκταθούν, να συγκολληθούν, να χρησιμοποιηθούν σε αναζητήσεις χαρακτήρων και επιμέρους strings.
- Οι επιμέρους χαρακτήρες ενός string είναι προσβάσιμοι μέσω του τελεστή [].
- Μπορούν να χρησιμοποιηθούν με τους (υπερφορτωμένους) λογικούς τελεστές: $<$ $<=$ $>$ $>=$ $==$ $!=$
- Ένα string μπορεί να αρχικοποιηθεί από έναν πίνακα char.
- Η μέθοδος `c_str()` ενός string μπορεί να χρησιμοποιηθεί για να ληφθεί το αντίστοιχο C-string, μια που πολλές υπάρχουσες βιβλιοθήκες παρέχουν συναρτήσεις που δέχονται ορίσματα C-strings.

C++ strings

- Η γενίκευση των C-strings (αλφαριθμητικών ακολουθιών) οδηγεί στην κλάση της C++ που ονομάζεται `string`.
- Τα C++ strings μπορούν να συγκριθούν, να επεκταθούν, να συγκολληθούν, να χρησιμοποιηθούν σε αναζητήσεις χαρακτήρων και επιμέρους strings.
- Οι επιμέρους χαρακτήρες ενός string είναι προσβάσιμοι μέσω του τελεστή [].
- Μπορούν να χρησιμοποιηθούν με τους (υπερφορτωμένους) λογικούς τελεστές: $<$ $<=$ $>$ $>=$ $==$ $!=$
- Ένα string μπορεί να αρχικοποιηθεί από έναν πίνακα char.
- Η μέθοδος `c_str()` ενός string μπορεί να χρησιμοποιηθεί για να ληφθεί το αντίστοιχο C-string, μια που πολλές υπάρχουσες βιβλιοθήκες παρέχουν συναρτήσεις που δέχονται ορίσματα C-strings.
- Η μέθοδος `size()` ενός string δίνει το μήκος του σε byte, λαμβάνοντας υπόψη τους όποιους multi-byte χαρακτήρες (π.χ. τους ελληνικούς Unicode).

C++ strings

Παραδείγματα:

```
string str1 ;   // Εξ ορισμού μέθοδος κατασκευής, κενό string
str1 = "Τμήμα"; // έμμεση απόδοση τιμής;
```

C++ strings

Παραδείγματα:

```
string str1 ;   // Εξ ορισμού μέθοδος κατασκευής, κενό string
str1 = "Τμήμα"; // έμμεση απόδοση τιμής;
```

```
string str2("Φυσικής") ;  // κατασκευή με αρχικοποίηση
```

C++ strings

Παραδείγματα:

```
string str1 ;   // Εξ ορισμού μέθοδος κατασκευής, κενό string
str1 = "Τμήμα"; // έμμεση απόδοση τιμής;
```

```
string str2("Φυσικής") ;  // κατασκευή με αρχικοποίηση
```

```
string str3 = "Γιάννης" ;    // κατασκευή και έμμεση απόδοση τιμής
if ( str3 == "Γιάννης" ) {   // έλεγχος ισότητας
  cout << "Γεια σου Γιάννη..." << endl;
}
```

C++ strings

Παραδείγματα:

```
string str1 ;   // Εξ ορισμού μέθοδος κατασκευής, κενό string
str1 = "Τμήμα"; // έμμεση απόδοση τιμής;
```

```
string str2("Φυσικής") ;  // κατασκευή με αρχικοποίηση
```

```
string str3 = "Γιάννης" ;    // κατασκευή και έμμεση απόδοση τιμής
if ( str3 == "Γιάννης" ) {   // έλεγχος ισότητας
  cout << "Γεια σου Γιάννη..." << endl;
}
```

```
string str4 = str1 + " " + str2 ; // συγκόλληση μέσω του τελεστή +
cout << str4 << endl;             // τυπώνει στην οθόνη "Τμήμα Φυσικής"
```

C++ strings

Παραδείγματα:

```
string str1 ;   // Εξ ορισμού μέθοδος κατασκευής, κενό string
str1 = "Τμήμα"; // έμμεση απόδοση τιμής;
```

```
string str2("Φυσικής") ;  // κατασκευή με αρχικοποίηση
```

```
string str3 = "Γιάννης" ;    // κατασκευή και έμμεση απόδοση τιμής
if ( str3 == "Γιάννης" ) {   // έλεγχος ισότητας
  cout << "Γεια σου Γιάννη..." << endl;
}
```

```
string str4 = str1 + " " + str2 ; // συγκόλληση μέσω του τελεστή +
cout << str4 << endl;             // τυπώνει στην οθόνη "Τμήμα Φυσικής"
```

```
cout << str4.size() << endl;  // τυπώνει στην οθόνη 25, επειδή οι 12 ελληνικοί
                              // χαρακτήρες είναι 2-byte Unicode, και το κενό
                              // μεταξύ των λέξεων απασχολεί 1 byte
```

Αρχεία στη C++

Αρχεία στη C++ (επικεφαλίδα <fstream>) : εγγραφή

- Η **εγγραφή δεδομένων** σε ένα αρχείο εξόδου,
  γίνεται κατ' αναλογία της εγγραφής στο ρεύμα εξόδου cout.

Αρχεία στη C++ (επικεφαλίδα <fstream>) : εγγραφή

- Η **εγγραφή δεδομένων** σε ένα αρχείο εξόδου,
  γίνεται κατ' αναλογία της εγγραφής στο ρεύμα εξόδου cout.
- Για τον ορισμό ενός ρεύματος εξόδου σε αρχείο, χρησιμοποιείται η κλάση `ofstream`,
  από την επικεφαλίδα `<fstream>`, όπως φαίνεται στο ακόλουθο παράδειγμα:

Αρχεία στη C++ (επικεφαλίδα <fstream>) : εγγραφή

- Η **εγγραφή δεδομένων** σε ένα αρχείο εξόδου,
  γίνεται κατ' αναλογία της εγγραφής στο ρεύμα εξόδου cout.
- Για τον ορισμό ενός ρεύματος εξόδου σε αρχείο, χρησιμοποιείται η κλάση `ofstream`, από την επικεφαλίδα `<fstream>`, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   ofstream F_out("myData.dat");   // Δημιουργία αρχείου εξόδου
   for ( int i=10 ; i<15 ; i++ ) {
     // εγγραφή στο αρχείο
     F_out << i << " " << i*i << " " << i*i*i << endl;
   }
}
```

Αρχεία στη C++ (επικεφαλίδα <fstream>) : εγγραφή

- Η **εγγραφή δεδομένων** σε ένα αρχείο εξόδου,
  γίνεται κατ' αναλογία της εγγραφής στο ρεύμα εξόδου cout.
- Για τον ορισμό ενός ρεύματος εξόδου σε αρχείο, χρησιμοποιείται η κλάση `ofstream`, από την επικεφαλίδα `<fstream>`, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   ofstream F_out("myData.dat");   // Δημιουργία αρχείου εξόδου
   for ( int i=10 ; i<15 ; i++ ) {
     // εγγραφή στο αρχείο
     F_out << i << " " << i*i << " " << i*i*i << endl;
   }
}
```

- Μεταγλώττιση, Εκτέλεση, Εμφάνιση περιεχομένων του αρχείου myData.dat

```
[bash 1]   g++  -std=c++11  prog1.cpp  -o prog1.exe
[bash 2]   ./prog1.exe
[bash 3]   cat myData.dat
10 100 1000
11 121 1331
12 144 1728
13 169 2197
14 196 2744
```

Αρχεία στη C++ (επικεφαλίδα <fstream>) : ανάγνωση

- Η **ανάγνωση δεδομένων** από ένα αρχείο εισόδου,
  γίνεται κατ' αναλογία της ανάγνωσης από το ρεύμα εισόδου cin.

Αρχεία στη C++ (επικεφαλίδα <fstream>) : ανάγνωση

- Η **ανάγνωση δεδομένων** από ένα αρχείο εισόδου, γίνεται κατ' αναλογία της ανάγνωσης από το ρεύμα εισόδου cin.
- Για τον ορισμό ενός ρεύματος εισόδου από αρχείο, χρησιμοποιείται η κλάση `ifstream`, από την επικεφαλίδα `<fstream>`. Στο ακόλουθο παράδειγμα, διαβάζονται τα περιεχόμενα του αρχείου myData.dat που εγγράφηκε προηγουμένως.

Αρχεία στη C++ (επικεφαλίδα <fstream>) : ανάγνωση

- Η **ανάγνωση δεδομένων** από ένα αρχείο εισόδου, γίνεται κατ' αναλογία της ανάγνωσης από το ρεύμα εισόδου cin.
- Για τον ορισμό ενός ρεύματος εισόδου από αρχείο, χρησιμοποιείται η κλάση `ifstream`, από την επικεφαλίδα `<fstream>`. Στο ακόλουθο παράδειγμα, διαβάζονται τα περιεχόμενα του αρχείου myData.dat που εγγράφηκε προηγουμένως.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  int i1, i2, i3;
  ifstream F_in("myData.dat");          // άνοιγμα αρχείου εισόδου
  for ( int i=10 ; i<15 ; i++ ) {
    F_in >> i1 >> i2 >> i3 ;            // ανάγνωση από το αρχείο
    cout << i3 << " " << i1 << endl;
  }
}
```

Αρχεία στη C++ (επικεφαλίδα <fstream>) : ανάγνωση

- Η **ανάγνωση δεδομένων** από ένα αρχείο εισόδου, γίνεται κατ' αναλογία της ανάγνωσης από το ρεύμα εισόδου cin.
- Για τον ορισμό ενός ρεύματος εισόδου από αρχείο, χρησιμοποιείται η κλάση `ifstream`, από την επικεφαλίδα `<fstream>`. Στο ακόλουθο παράδειγμα, διαβάζονται τα περιεχόμενα του αρχείου myData.dat που εγγράφηκε προηγουμένως.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  int i1, i2, i3;
  ifstream F_in("myData.dat");          // άνοιγμα αρχείου εισόδου
  for ( int i=10 ; i<15 ; i++ ) {
    F_in >> i1 >> i2 >> i3 ;            // ανάγνωση από το αρχείο
    cout << i3 << " " << i1 << endl;
  }
}
```

- Μεταγλώττιση, Εκτέλεση

```
[bash 1]   g++  -std=c++11  prog2.cpp  -o prog2.exe
[bash 2]   ./prog2.exe
1000 10
1331 11
1728 12
2197 13
2744 14
```