ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Τεχνολογίες Υλοποίησης Αλγορίθμων

Χρήστος Ζαρολιάγκης

Καθηγητής
Τμήμα Μηχ/κων Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
email: zaro@ceid.upatras.gr

Εισαγωγή στην C++ - 2

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο **«Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών»** έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Περιεχόμενα

Πίνακες και Διανύσματα

type array_name (size_d1)(size_d2) ...

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

int main()
{
    const number_of_students = 30;
    float sum = 0, average = 0;
    float marks[number_of_students];
    int student;
    cout << "Type in the students' marks, one per line";
    cout << endl;

    for (student=0; student<number_of_students; ++student)
    {
        cin >> marks[student];
        sum += marks[student];
    }

    average = sum/number_of_students;
    cout << "The average mark is: " << average << endl;
    return 0;
}
```

Αριθμοδείκτης Πίνακα (σε κάθε διάσταση d_i):

```
0 .. size_di - 1
```

Πίνακες

– Οι πίνακες αποθηκεύονται *κατά γραμμές*.

– Αρχικοποίηση:

```
int a[2][3] = {{1,2,3},{4,5,6}};
int b[2][3] = {1,2,3,4,5};
int c[2][3] = {{1,2},{4}};
```

Τιμές του a κατά γραμμές:

1	2	3
4	5	6

Τιμές του b κατά γραμμές:

1	2	3
4	5	0

Τιμές του c κατά γραμμές:

1	2	0
4	0	0

Διανύσματα (Vectors)

- Εναλλακτική αναπαράσταση στον ενσωματωμένο τύπο πίνακα.
- Περιέχουσα κλάση (container class): μπορεί να περιέχει αντικείμενα κάθε τύπου.
- Διαθέτει έλεγχο ορίων αριθμοδείκτη.

– Αρχείο κεφαλίδας: <vector>

– Αρχικοποίηση:

```
vector<int> v1(10); // equivalent to int i[10]
vector<int> v2;
```

– Αριθμοδείκτης όπως και στον τύπο πίνακα:

```
void simple_example()
{
    const int elem_size=10;
    vector<int> ivec(elem_size);
    int ia[elem_size];

    for (int ix=0; ix<elem_size; ++ix)
        ia[ix] = ivec[ix];
    . . .
}
```

Έλεγχος κενότητας ή μεγέθους

```
if (ivec.empty())
    // do something
for (int ix=0; ix<ivec.size(); ++ix)
    cout << ivec[ix] << endl;
. . .
```

– Αρχικοποίηση διανύσματος (συνέχεια):

```
vector<int> ivec(10,2); // all elements initialised to 2

int ia[5] = {-2, 50, 0, 37, 686};
vector<int> ivec(ia, ia+5); // [first, last)
                            // initialises ivec with
                            // a copy of the elements of ia
vector<string> svec;
void init_and_assign()
{
  vector<string> user_names(svec); // initialises one
                                   // vector with another
  . . .
  svec = user_names; // copies one vector into another
  . . .
}
```

Εντολές επιλογής if και if/else

```
if (expression)
{
    statement-1;
    statement-2;
    . . .
}
if (expression)
{
    i-statement-1;
    i-statement-2;
    . . .
}
else
{
    e-statement-1;
    e-statement-2;
    . . .
}
if (expression)
{
    i-statement-1; . . .
}
else if (expression)
{
    e1-statement-1; . . .
}
else
{
    //...
}
```

Παράδειγμα

```
if (grade >= 5)
  cout << "Passed" << endl;
else
{
  cout << "Failed" << endl;
  cout << "You must take this course again" << endl;
}
```

Η εντολή πολλαπλής επιλογής switch

```
char grade;
int aCount=0, bCount=0, cCount=0, dCount=0;
// ...
switch (grade)
{
    case 'A': case 'a':
        ++aCount;
        break;          // important !
    case 'B': case 'b':
        ++bCount;
        break;
    case 'C': case 'c':
        ++cCount;
        break;
    case 'D': case 'd':
        ++dCount;
        break;
    default:
        cout << "Incorrect letter grade entered" << endl;
        break; // not necessary, but recommended
 }
```

Βρόχοι επανάληψης

Ο βρόχος for

```
for (init-statement; expression 1; expression 2)
{
    statement-1;
    statement-2;
}
```

Ο βρόχος while

```
while (expression)
{
    statement-1;
    statement-2;
}
```

Ο βρόχος do-while

```
do {
    statement-1;
    statement-2;
} while (expression);
```

Break και Continue

- Και οι δύο αλλάζουν τη ροή ελέγχου.
- Η break (χρησιμοποιείται σε εντολές while, for, do/while, switch) προκαλεί την άμεση έξοδο από τον κορμό της εντολής.
- Η continue (χρησιμοποιείται σε εντολές while, for, do/while) προκαλεί την παράλειψη της εκτέλεσης των υπόλοιπων εντολών στον κορμό του βρόχου και τη συνέχιση με την επόμενη επανάληψη του βρόχου.
  - while, do/while : η συνθήκη τερματισμού του βρόχου εξετάζεται αμέσως μετα την continue.
  - for : πρώτα εκτελείται η παράσταση αύξησης και μετά εξετάζεται η συνθήκη τερματισμού του βρόχου.

Παράδειγμα

```
#include <iostream>
using std::cout;
using std::endl;

int main()
{
  int i,j;
  for (i=1; i<=10; i++)
  {
    if (i==5)
      break; // break loop when i == 5
    cout << i << " ";
  }
  cout << endl << "Broke out of loop at i=" << i << endl;
  for (j=1; j<=10; j++)
  {
    if (j==5)
      continue; // skip remaining code in the
                // for-loop when j == 5
    cout << j << " ";
  }
  cout << endl << "Used continue to skip printing";
  cout << " the value 5" << endl;
  return 0;
}
```

Δείκτης (pointer):

- Μεταβλητή η οποία περιέχει σαν τιμή μια διεύθυνση μνήμης.
- Χρησιμοποιείται για να αποθηκεύσει τη διεύθυνση μιας μεταβλητής που περιέχει μια συγκεκριμένη τιμή.

```
int * ptr, count; // ptr is a pointer to
                  // an integer value
```

Τελεστές Δεικτών

* (αποαναφοροποίηση), & (διεύθυνση μνήμης)


```
int i=37;
int * iPtr;
iPtr = &i; // assigns the address of i to
           // the pointer variable iPtr

// The preceding two lines could be
// alternatively written as
// int * iPtr = &i;

cout << *iPtr << endl; // * dereferences iPtr;
                       // this statement prints
                       // the value of i, i.e., 37
```

Αναφορά (reference)

- Είναι ένα εναλλακτικό όνομα για ένα αντικείμενο.
- Επιτρέπει τον έμμεσο χειρισμό ενός αντικειμένου.

```
int val = 516;
int & refval = val;
int & refval2;          // wrong! A reference must always
                        // be initialised to an object
int & refval = &val; // wrong! refval is of type int
                        // not int*
```

- Εφόσον οριστεί, δεν μπορεί να αποτελέσει αναφορά ενός άλλου αντικειμένου.

```
int min_val = 0;

// it sets val to the value of min_val
// refval is not set to refer to min_val
refval = min_val;
```

- Όλες οι λειτουργίες σε μια αναφορά ουσιαστικά εφαρμόζονται στο αντικείμενο στο οποίο αναφέρεται.

```
refval += 2;            // adds 2 to val
int i = refval;         // assigns to i the current
                        // value of val
int * ptr = &refval; // initialises ptr with
                        // the address of val
```

Παράδειγμα 1

```
#include <iostream>
using namespace std;
int main()
{ int i = 333;
  int * p = &i;
  cout << "i           : " << i << endl;
  cout << "address of i: " << &i << endl;
  cout << "value of p  : " << p << endl;
  cout << "value of &p : " << &p << endl;
  cout << "value of *p : " << *p << endl;
  cout << "value of *&p: " << *&p << endl << endl;

  *p = 999;
  cout << "i           : " << i << endl;
  cout << "address of i: " << &i << endl;
  cout << "value of p  : " << p << endl;
  cout << "value of &p : " << &p << endl;
  cout << "value of *p : " << *p << endl;
  cout << "value of *&p: " << *&p << endl;
  return 0;
}
```

Έξοδος από Diogenis:

```
i           : 333          i           : 999
address of i: 0xffbef8fc   address of i: 0xffbef8fc
value of p  : 0xffbef8fc   value of p  : 0xffbef8fc
value of &p : 0xffbef8f8   value of &p : 0xffbef8f8
value of *p : 333          value of *p : 999
value of *&p: 0xffbef8fc   value of *&p: 0xffbef8fc
```

Παράδειγμα 2

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

int main()
{
    int i;
    float sum=0.0, a[10], * pstart, * pend;

    for (i=0; i<10; i++)
    {
        a[i]=i; sum += a[i];
    }
    cout << "Sum: " << sum << endl;

    sum=0.0;
    pstart = &a[0]; // or pstart = a;
    pend = pstart + 10;

    while (pstart < pend)
    {
        sum += *pstart++;
    }
    cout << "Sum: " << sum << endl;
    return 0;
}
```

Συναρτήσεις (Functions)

- Ομαδοποιεί τμήματα ενός προγράμματος (π.χ., παραστάσεις, εντολές) για να διευκολύνει την επαναχρησιμοποίηση αυτών των τμημάτων μέσα στο πρόγραμμα.
- Αποτελείται από:
  - το όνομα της συνάρτησης
  - τη λίστα παραμέτρων (τυπικές παράμετροι) (είσοδος)
  - τον τύπο τιμής που επιστρέφει (έξοδος)
  - τον κορμό (κώδικας που υλοποιεί την συνάρτηση)
- *Ορισμός συνάρτησης:* τύπος τιμής επιστροφής ακολουθούμενος από το όνομα, τη λίστα των παραμέτρων και τον κορμό της συνάρτησης.
- *Δήλωση συνάρτησης:* τύπος τιμής επιστροφής ακολουθούμενος από το όνομα και τη λίστα των παραμέτρων (δηλαδή μόνο η επικεφαλίδα της συνάρτησης).

Παραδείγματα ορισμών συναρτήσεων

```
// return the absolute value of iobj
int abs(int iobj)
{
    return( iobj < 0 ? -iobj : iobj );
}

// return the smaller of v1 and v2
int min(int v1, int v2)
{
    return( v1 < v2 ? v1 : v2 );
}

// return the greatest common divisor
int gcd(int v1, int v2)
{
    while ( v2 )
    {
        int temp = v2;
        v2 = v1 % v2;
        v1 = temp;
    }
    return v1;
}
```

Κλήση συνάρτησης

- *Κλήση συνάρτησης:* όνομα της συνάρτησης ακολουθούμενο από τον *τελεστή κλήσης* () και παρέχοντας *πραγματικές παραμέτρους* (*actual parameters*) ή *ορίσματα* (*arguments*).

Π.χ.: `min(i,j); gcd(35,49);`

Πλήρες Παράδειγμα – 1

```
#include <iostream>

// return the absolute value of iobj
int abs(int iobj)
{
    return( iobj < 0 ? -iobj : iobj );
}

// return the smaller of v1 and v2
int min(int v1, int v2)
{
    return( v1 < v2 ? v1 : v2 );
}

// return the greatest common divisor
int gcd(int v1, int v2)
{
    while ( v2 )
    {
        int temp = v2;
        v2 = v1 % v2;
        v1 = temp;
    }
    return v1;
}
```

Πλήρες Παράδειγμα – 1 (συνέχεια)

```
int main()
{
    cout << "Enter first value: ";
    int i;
    cin >> i;

    cout << "Enter second value: ";
    int j;
    cin >> j;

    cout << endl << "min: " << min(i, j) << endl;
    i = abs(i);
    j = abs(j);
    cout << "gcd: " << gcd(i, j) << endl;
    return 0;
}
```

Συναρτήσεις

- Οι συναρτήσεις καλούνται κατά το χρόνο εκτέλεσης (run-time) του προγράμματος, εκτός αν έχουν δηλωθεί inline (εμβόλιμες).
- Εμβόλιμες (inline) συναρτήσεις: ο κορμός της συνάρτησης εισάγεται στο σημείο κλήσης της κατά τη διάρκεια της μεταγλώττισης (αν ο μεταγλωττιστής θεωρήσει ότι αυτό είναι δυνατόν).
- Μια συνάρτηση πρέπει να *δηλωθεί* πριν κληθεί.
  - Ο ορισμός της συνάρτησης χρησιμεύει και ως *δήλωση* της.
  - Όμως,
    - Μια συνάρτηση μπορεί να ορισθεί μόνο μία φορά σε ένα πρόγραμμα, ενώ μπορεί να δηλωθεί πολλές φορές.
    - Η C++ παρέχει μηχανισμούς που επιτρέπουν:
      (1) Μία συνάρτηση να χρησιμοποιείται σε αρχεία διαφορετικά από αυτό που περιέχει τον ορισμό της.
      (2) Τον διαχωρισμό μεταξύ προδιαγραφών (ή αλλιώς της *διασύνδεσης*) μιας συνάρτησης και της υλοποίησής της.

Πλήρες Παράδειγμα – 2

```
#include <iostream>

// Function declarations
int abs(int);
int min(int , int);
int gcd(int , int);

// Function definitions

// return the absolute value of iobj
inline int abs(int iobj)
{
    return( iobj < 0 ? -iobj : iobj );
}

// return the smaller of v1 and v2
inline int min(int v1, int v2)
{
    return( v1 < v2 ? v1 : v2 );
}

// return the greatest common divisor
int gcd(int v1, int v2)
{
    while ( v2 )
    {
        int temp = v2;
        v2 = v1 % v2;
        v1 = temp;
    }
    return v1;
}
```

Πλήρες Παράδειγμα – 2 (συνέχεια)

```
int main()
{
    cout << "Enter first value: ";
    int i;
    cin >> i;

    cout << "Enter second value: ";
    int j;
    cin >> j;

    cout << endl << "min: " << min(i, j) << endl;
    i = abs(i);
    j = abs(j);
    cout << "gcd: " << gcd(i, j) << endl;
    return 0;
}
```

Δηλώσεις Συναρτήσεων

- Δηλώσεις συναρτήσεων (και ορισμοί inline συναρτήσεων) συνιστάται να περιέχονται σε *αρχεία κεφαλίδων (.h – header files)*
- Ορισμοί συναρτήσεων συνιστάται να περιέχονται σε ένα *ξεχωριστό* συσχετιζόμενο .C αρχείο.

Πλήρες Παράδειγμα – 3

```
// localmath.h
#ifndef LOCALMATH_H
#define LOCALMATH_H

// return the absolute value of iobj
inline int abs(int iobj)
{
    return( iobj < 0 ? -iobj : iobj );
}

// return the smaller of v1 and v2
inline int min(int v1, int v2)
{
    return( v1 < v2 ? v1 : v2 );
}

int gcd(int , int);

#endif
```

Πλήρες Παράδειγμα – 3 (συνέχεια)

```
// localmath.C

#include "localmath.h"

// return the greatest common divisor
int gcd(int v1, int v2)
{
    while ( v2 )
    {
        int temp = v2;
        v2 = v1 % v2;
        v1 = temp;
    }
    return v1;
}
```

Πλήρες Παράδειγμα – 3 (συνέχεια)

```
// fun-main.C
#include <iostream>
#include "localmath.h"
using namespace std;

int main()
{
    cout << "Enter first value: ";
    int i;
    cin >> i;

    cout << "Enter second value: ";
    int j;
    cin >> j;

    cout << endl << "min: " << min(i, j) << endl;
    i = abs(i);
    j = abs(j);
    cout << "gcd: " << gcd(i, j) << endl;
    return 0;
}
```

Μεταγλώττιση & Σύνδεση

- Μεταγλώττιση:

```
g++ -c localmath.C
g++ -c fun-main.C
```

- Σύνδεση:

```
g++ -o fun-main fun-main.o localmath.o
```

Μεταβίβαση Ορισμάτων (Argument Passing)

Η C++ είναι *αυστηρή* στον έλεγχο τύπων *(strongly-typed)*:

> Ο τύπος κάθε πραγματικού ορίσματος συγκρίνεται με τον τύπο της αντίστοιχης τυπικής παραμέτρου.
> Σε περίπτωση ασυμφωνίας, πραγματοποιείται προαγωγή τύπου εφόσον είναι δυνατή. Διαφορετικά, εμφανίζεται λάθος κατά το χρόνο μεταγλώπτισης (compile-time error).

- **Μέθοδοι Μεταβίβασης Ορισμάτων**

  - μεταβίβαση κατ' αξία (pass-by-value)
  - μεταβίβαση κατ' αναφορά (pass-by-reference)

Μεταβίβαση κατ' αξία

- Προεπιλεγμένη (default) μέθοδος.
- Δημιουργείται ένα *αντίγραφο* της τιμής του ορίσματος και μεταβιβάζεται στον τοπικό χώρο αποθήκευσης της καλούμενης συνάρτησης.
- Το περιεχόμενο των ορισμάτων δεν αλλάζει.

Παράδειγμα:

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

// swap() does not swap the value of the arguments!
void swap(int v1, int v2)
{
    int tmp = v2; v2 = v1; v1 = tmp;
}

int main()
{
    int i = 10; int j = 20;

    cout << "Before swap():\t i: "
         << i << "\t j: " << j << endl;
    swap(i,j);
    cout << "After swap():\t i: "
         << i << "\t j: " << j << endl;
    return 0;
}
```

Μεταβίβαση κατ΄ αξία

Η μεταβίβαση κατ΄ αξία δεν είναι κατάλληλη για κάθε περίπτωση. Για παράδειγμα:

- όταν ένα μεγάλο αντικείμενο πρέπει να μεταβιβαστεί σαν όρισμα
- όταν οι τιμές των ορισμάτων πρέπει να τροποποιηθούν

Μεταβίβαση κατ΄ αναφορά:

- Δίνει τη δυνατότητα στην καλούμενη συνάρτηση να προσπελάσει άμεσα τα δεδομένα της καλούσας συνάρτησης και να τα τροποποιήσει.
- Μπορεί να επιτευχθεί χρησιμοποιώντας παραμέτρους τύπου δείκτη ή αναφοράς (η συνήθης υλοποίηση στον μεταγλωττιστή είναι η μεταβίβαση της *διεύθυνσης* των ορισμάτων).

Παράδειγμα

```
// pswap() swaps the values that v1 and v2 address
void pswap(int * v1, int * v2)
{
    int tmp = *v2;
    *v2 = *v1;
    *v1 = tmp;
}
```

Κλήση της συνάρτησης: `pswap(&i,&j);`

```
// rswap() swaps the values that v1 and v2 refer
void rswap(int & v1, int & v2)
{
    int tmp = v2;
    v2 = v1;
    v1 = tmp;
}
```

Κλήση της συνάρτησης: `rswap(i,j);`

Παράμετροι Τύπου Αναφοράς και Δείκτη

▷ *Σε ποιες περιπτώσεις προσδιορίζουμε μια παράμετρο σαν αναφορά·*

- Όταν δεν θέλουμε να αλλάξουμε τις παραμέτρους σε δείκτες για να επιτρέψουμε τροποποίηση των τιμών των ορισμάτων.
- Όταν θέλουμε να επιστρέψουμε επιπρόσθετα αποτελέσματα.
- Όταν θέλουμε να μεταβιβάσουμε μεγάλα αντικείμενα (επίσης σωστό και για παραμέτρους τύπου δείκτη).

▷ *Όταν μια παράμετρος δεν πρόκειται να τροποποιηθεί, τότε είναι καλύτερο να δηλωθεί σαν μία αναφορά σε έναν τύπο const, δηλ. const T & a.*

▷ *Επιλογή μεταξύ παραμέτρων τύπου αναφοράς και δείκτη*

- Όταν μια παράμετρος μπορεί να αναφέρεται σε διαφορετικά αντικείμενα μέσα σε μια συνάρτηση ή σε κανένα αντικείμενο, τότε μια παράμετρος τύπου δείκτη πρέπει να χρησιμοποιηθεί.
- Σε όλες τις άλλες περιπτώσεις συνιστάται η χρήση παραμέτρων τύπου αναφοράς.

Μεταβίβαση Πινάκων σαν Ορίσματα

Οι πίνακες στην C++ δεν μεταβιβάζονται ποτέ κατ' αξία. Ένας πίνακας μεταβιβάζεται σαν ένας δείκτης στο πρώτο του στοιχείο. Οι ακόλουθοι ορισμοί συναρτήσεων είναι όλοι ισοδύναμοι:

```
int array_sum_1(int x[], int n)
{
    int i, t=0;
    for (i=0; i<n; i++)
        t += x[i];
    return t;
}
int array_sum_2(int x[10], int n)
{
    int i, t=0;
    for (i=0; i<n; i++)
        t += x[i];
    return t;
}
int array_sum_3(int * x, int n)
{
    int i, t=0;
    for (i=0; i<n; i++)
        t += x[i];
    return t;
}
```

Μεταβίβαση Πινάκων σαν Ορίσματα

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

int main()
{
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    int sa=10;

    cout << "Sum 1: " << array_sum_1(a,sa) << endl;
    cout << "Sum 2: " << array_sum_2(a,sa) << endl;
    cout << "Sum 3: " << array_sum_3(a,sa) << endl;
    return 0;
}
```

Πολυδιάστατοι πίνακες:

- Όλες οι διαστάσεις, εκτός από την πρώτη, πρέπει να είναι γνωστές κατά το χρόνο μεταγλώπτισης (compile-time).
- Αλλιώς, πρέπει να χρησιμοποιηθεί η μέθοδος 3.

Επιστροφή Τιμών Συναρτήσεων

- *Επιστροφή αξίας (value return):* η καλούσα συνάρτηση λαμβάνει ένα αντίγραφο της τιμής που επιστρέφει η καλούμενη συνάρτηση.
- *Επιστροφή αναφοράς ή δείκτη (reference or pointer return):* η καλούσα συνάρτηση λαμβάνει την lvalue ή τη διεύθυνση που επιστρέφει η καλούμενη συνάρτηση.
  Π.χ. `T & Fun(int i, int & j)`
- *const επιστροφή αναφοράς:* παρόμοια με την επιστροφή αναφοράς με τη διαφορά ότι το αντικείμενο που επιστρέφεται ορίζεται ως σταθερό αντικείμενο.
  Π.χ. `const T & Fun(int i, int & j)`
- *Καμία επιστροφή τιμής:* τύπος επιστροφής void.

Αρχέτυπα (Templates)

- Θυμηθείτε τις συναρτήσεις swap() που ανταλλάσσουν int τιμές. Τι πρέπει να κάνουμε αν θέλουμε να ανταλλάξουμε τιμές διαφορετικού τύπου, π.χ. float, long, string ;
- **Αρχέτυπα:** προσφέρουν έναν μηχανισμό για *παραμετροποίηση* τύπων μέσα σε μια συνάρτηση ή στον ορισμό μιας κλάσης. Αυτές οι παράμετροι χρησιμοποιούνται σαν θέσεις γενικών τύπων σε ουσιαστικά αμετάβλητο (ως προς συγκεκριμένους τύπους) κώδικα.

  Παράδειγμα:

```
// rswap() swaps the values that v1 and v2 refer
template<typename T> // could be also template<class T>
void rswap(T & v1, T & v2)
{
    T tmp = v2;
    v2 = v1;
    v1 = tmp;
}
```

Γενική Δήλωση Αρχετύπου

- **Γενική Δήλωση Αρχετύπου**

```
template<typename T1, typename T2, ...,
         typename Tn>
T1 fun(T2, T3, ..., Tn);
```

(Αντί για `typename` μπορεί να γραφεί και το `class`)

Ένα Πλήρες Παράδειγμα

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

template<typename T>
T min(T a, T b)
{
    return a<b ? a : b;
}

int main()
{
    int i=100, j=200;
    float a=23.75, b=8.45;
    string s("data"), t("datum");

    // template instantiation: min of integers
    cout << "min(i,j)=" << min(i,j) << endl;

    // template instantiation: min of floats
    cout << "min(a,b)=" << min(a,b) << endl;

    // template instantiation: min of strings
    cout << "min(s,t)=" << min(s,t) << endl;
    return 0;
}
```

Εξειδίκευση αρχετύπου (template specialization)

Αν η γενικευμένη υλοποίηση δεν είναι (για οποιοδήποτε λόγο π.χ. απόδοση) κατάλληλη για έναν συγκεκριμένο τύπο, είναι δυνατόν να οριστεί υλοποίηση μόνο γι αυτόν τον τύπο.

Παράδειγμα - εξειδίκευση της swap() για τον τύπο int :

```
template<>
void swap(int & v1, int & v2)
{
    int temp = v1;
    v1 = v2;
    v2 = temp;
}
```

Εμβέλεια (Scope)

Το τμήμα του προγράμματος μέσα στο οποίο ένα αντικείμενο, μια συνάρτηση, ένας τύπος, ή ένα αρχέτυπο μπορεί να χρησιμοποιηθεί.

Τοπική εμβέλεια: τμήμα προγράμματος που περιέχεται μέσα στον ορισμό μιας συνάρτησης. Μέσα σε μια συνάρτηση, ένα μπλοκ ({ }) αναπαριστάνει επίσης μια ξεχωριστή τοπική εμβέλεια.

Εμβέλεια χώρου ονομάτων: τμήμα προγράμματος που δεν περιέχεται μέσα σε δήλωση συνάρτησης, ορισμό συνάρτησης, ή ορισμό κλάσης. Η πιο εξωτερική εμβέλεια χώρου ονομάτων καλείται *καθολική εμβέλεια*.

Εμβέλεια κλάσης: τμήμα προγράμματος που περιέχεται μέσα στον ορισμό μιας κλάσης (θα συζητηθεί αργότερα).

Κατανομή μνήμης (Memory allocation)

- **Στατική:** η κατανομή γίνεται από τον μεταγλωττιστή καθώς επεξεργάζεται τον πηγαίο κώδικα.
- **Δυναμική:** η κατανομή γίνεται από μια βιβλιοθήκη που καλείται κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Trade-off: απόδοση (efficiency) vs. ευελιξία (flexibility)

Στατική κατανομή μνήμης

- **Αυτόματη:** ένα αντικείμενο κατανέμεται όταν η εκτέλεση του προγράμματος φτάσει στο μπλοκ μέσα στο οποίο ορίζεται, διατηρείται όσο χρόνο το μπλοκ είναι ενεργό, και αποδεσμεύεται όταν η εκτέλεση του προγράμματος αφήνει το μπλοκ.

  float x; // automatic

- **Στατική:** ένα στατικό αντικείμενο κατανέμεται από το στιγμή που το πρόγραμμα αρχίζει την εκτέλεσή του και διατηρείται καθ' όλη τη διάρκεια ζωής του προγράμματος.

```
// used for global variables
// and function names
extern float x;
// used for local variables in functions;
// retain their values when the function
// is exited
static int i=1;
```

Παράδειγμα

```
#include <iostream>
using namespace std;
int x=1;  // global variable

void a(void)
{
    int x=25; // initialised every time a is called
    cout << x << endl;
    ++x;
    cout << x << endl;
}

void b(void)
{
    static x=50; // initialised only the first time
                 // b is called
    cout << x << endl;
    ++x;
    cout << x << endl;}

void c(void)
{
    cout << x << endl;
    x *= 10;
    cout << x << endl;
}
```

Παράδειγμα (συνέχεια)

```
int main();
{
    int x=5; // local variable to main
    cout << x << endl;      // prints 5
    {                       // start new scope
        int x=7;
        cout << x << endl; // prints 7
    }
    cout << x << endl;      // prints 5

    a(); // prints 25 and 26
    b(); // prints 50 and 51
    c(); // prints 1 and 10
    a(); // prints 25 and 26
    b(); // prints 51 and 52
    c(); // prints 10 and 100
    return 0;
}
```

Δυναμική Κατανομή Μνήμης

Βασικές διαφορές μεταξύ στατικής και δυναμικής κατανομής μνήμης:

- Τα στατικά αντικείμενα είναι *επώνυμες* μεταβλητές τις οποίες χειριζόμαστε με άμεσο τρόπο.
  Τα δυναμικά αντικείμενα είναι *ανώνυμες* μεταβλητές τις οποίες χειριζόμαστε έμμεσα μέσω δεικτών.
- Κατανομή/Αποδέσμευση στατικών αντικειμένων γίνεται αυτόματα από τον μεταγλωττιστή.
  Κατανομή/Αποδέσμευση δυναμικών αντικειμένων γίνεται ρητά από τον προγραμματιστή χρησιμοποιώντας τις παραστάσεις των τελεστών new και delete.

Κατανομή δυναμικών αντικειμένων

- ```
  int * p = new int(128);
  // int * p; p = new int(128);

  // allocates unnamed object of type int,
  // initialises its value to 128,
  // and returns its address
  ```

- ```
  float * x = new float[100];
  // float * x; x = new float[100];

  // allocates an array of 100 elements of
  // type float, and returns the address of
  // the first element x[0]
  ```

Αποδέσμευση δυναμικών αντικειμένων

- ```
  delete p; // deletes a single object
  ```

- ```
  delete [] x; // deletes an array of objects
  ```

Χειρισμός Εξαιρέσεων

Εξαίρεση: σφάλμα που μπορεί να συμβεί κατά το χρόνο εκτέλεσης ενός προγράμματος, π.χ. αριθμοδείκτης πίνακα εκτός ορίων.

Οι εξαιρέσεις *εγείρονται* (throw) από το σύστημα ή τον προγραμματιστή και *συλλαμβάνονται* (catch) χρησιμοποιώντας το μπλοκ try − catch.

Παράδειγμα

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

int main()
{
  int i=0;
  int j=0;
  cout << " Type in a number > 10 " ;
  cin  >> i;

  try
  {
    if (i <= 10)
      throw j; // throws an int object with value 0
    else
      cout << " Number was " << i << endl;
  }
  catch (int j)
  {
    cout << " Exception raised " << endl;
  }

  try
  {
    x = new long long int [n];
  }
  // (...) is the catch-all handler
  catch (...)
  {
    cout << "Out of Memory" << endl;
  }
}
```

Υπερφόρτωση Συναρτήσεων

Επιτρέπει σε περισσότερες από μία συναρτήσεις να χρησιμοποιούν το *ίδιο όνομα* με την προϋπόθεση ότι κάθε τέτοια συνάρτηση έχει μια μοναδική λίστα παραμέτρων είτε κατά αριθμό, είτε κατά τύπο παραμέτρων (Προσοχή: δεν αρκεί να είναι διαφορετικός μόνο ο τύπος τιμής επιστροφής).

```
// An overloaded set of min() functions
. . .

int    min(int, int);
int    min(const int * parray, int size);
int    min(const char * str);
char   min(string);
string min(string, string);
```

Τέλος Ενότητας

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση **1.0**.

Σημείωμα Ιστορικού Εκδόσεων Έργου

Σημείωμα Ιστορικού Εκδόσεων Έργου

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει :

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει) μαζί με τους συνοδευόμενους υπερσυνδέσμους