C++ fundamental building blocks

(χωρίς να αναφερθούμε, όμως, σε έννοιες του αντικειμενοστρεφούς προγραμματισμού)

ΕΛΛΑΔΑ 2008
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΔΕΚ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

(γεια σας και χαρά σας)

HELLO WORLD

Κύριο πρόγραμμα

```
//A Simple C++ program
#include <iostream>
using namespace std;

int main(){
  cout << "Hello world!" << endl;
  return 0;
}
```

Χρειαζόμαστε
(α) βιβλιοθήκες (π.χ., iostream for i/o)
(β) std namespace (to be explained later)

Χρειαζόμαστε τη main() ως την κεντρική συνάρτηση του προγράμματός μας

(εισαγωγικά)

ΕΙΣΟΔΟΣ / ΕΞΟΔΟΣ

Είσοδος / Έξοδος δεδομένων

- Στη C++ η βασική είσοδος και έξοδος γίνεται με τα ρεύματα εισόδου / εξόδου **cout** και **cin**, τα οποία είναι αντικείμενα τύπου istream και ostream αντίστοιχα
- Ο τελεστής << παίρνει μια (σύνθετη) έκφραση από το πρόγραμμά μας και την εισάγει στο cout, το οποίο με τη σειρά του την στέλνει στην οθόνη
- Ο τελεστής >> εξάγει δεδομένα από το cin (το οποίο τα παίρνει από το πληκτρολόγιο) και τα στέλνει στις μεταβλητές υποδοχής

- Η είσοδος / έξοδος της C υποστηρίζεται για λόγους συμβατότητας, επίσης

Απλή είσοδος / έξοδος: cout / cin

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

main(){
    cout << "Hello world!" << endl;
}
```

Ισοδύναμα:

```
cout << "Hello" << "world!\n";
cout << "Hello world!\n";
```

Είσοδος / Έξοδος δεδομένων

```
#include <iostream>
using namespace std;
int main(){

  int i;
  cin >> i;
  float f;
  cin >> f;
  char c;
  cin >> c;
  char buf[100];
  cin >> buf;

  cin >> i >> f >> buf;
}
```

Είσοδος / Έξοδος δεδομένων

```
#include <iostream>
using namespace std;
int main(){
  int i;
  cin >> i;
  cout << "i = ";
  cout << i;
  cout << "\n";

  float f;
  cin >> f;
  cout << "f = ";
  cout << f;
  cout << "\n";

  cin >> i >> f;
  cout << "i = " << i << "\n" << "f = " << f << "\n" ;
  cout << "i = " << i << endl << "f = " << f << endl;
}
```

(βασικοί τύποι, αρχικοποιήσεις, strings, const)

ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ

Βασικοί τύποι

```
int xInt;                       //4 bytes (σπάνια 2)
short int xShortInt;            //2 bytes
long int xLong;                 //4 bytes
float xFloat;                   //4 bytes
double xDouble;                 //8 bytes
char xChar;                     //1 byte

bool xBool;                     //1 byte
xBool = true; xBool = false;
```

Πίνακες

Integers

```
int dataArray[3];
```

C Strings

```
#include <cstring>
char firstName[6];
main(){
      strcpy (firstName, "eddie");
}
```

Δηλώσεις Μεταβλητών

- Στη C++ δηλώσεις μεταβλητών μπορούν να γίνουν παντού και όχι μόνο στην αρχή κάθε συνάρτησης (including main())

```
int main(){
   int x, y, z;
   if(…) {…}
   // ……..
   float f;
}
```

Δηλώσεις τοπικών μεταβλητών παντού

- Επιτρέπονται δηλώσεις μεταβλητών παντού μέσα στο πρόγραμμα.

```
void f ()
{
    int x = 0;
    cout << x;
    int y = x + 1;
    cout << y;
    for (int i=1; i < y; i++){
       cout << i;
     }
}
```

Η i ΔΕΝ ζει μετά το block του for

Αρχικοποίηση Μεταβλητών

```
int counter(0);              //equiv. to: int counter=0;
```

```
int myArray[4] = {1,2,3,4};   //equiv. to:
myArray[0] = 1;               //we start from 0
myArray[1] = 2; ...
//could also say
int myArray[] = {1,2,3,4};
```

```
//size: 6, length: 6 –remember the last '\0'
char firstName[] = "eddie";
//size: 50, length: 6
char firstName[50] = "eddie";
```

Πολυδιάστατοι Πίνακες

```
int myMatrix[2][4];

int myMatrix[2][4] = {
    {1,2,3,4},
    {5,6,7,8}
};
```

Const

Pay particular attention: frequently used for (input) parameter passing, as well as global variables!!

const

```
main(){
   int const SIZE = 20;

   //can be used to initialize an array
   char buffer[SIZE];

   //will NOT work – const variables do NOT change
   SIZE = 8;
}
```

Διαφορές const και #define

- Το `#define` αντικαθιστά την έκφραση με την ισοδύναμή της, μέσω του preprocessor (ΔΕΝ είναι μέρος της C++)
- Το `#define` είναι εξ' ορισμού global ορισμός, ενώ το `const` έχει δικό του scope.
- Η συντακτική ορθότητα μιας δήλωσης `const` ελέγχεται αμέσως, ενώ για το `#define` μόνο αφού γίνει η αντικατάσταση. Το όποιο λάθος εμφανίζεται στη γραμμή του προγράμματος και όχι στη δήλωση του `#define`.

- **Το `#define` είναι συχνή πηγή σύγχυσης και bugs, ιδίως όταν ο κώδικας σπάσει σε πολλά αρχεία => θα αποφεύγουμε τα `#define` και θα χρησιμοποιούμε const μεταβλητές!**

const

- Πρακτικός κανόνας: όταν δηλώνω μια μεταβλητή σαν const ΔΕΝ μπορώ να αλλάξω οτιδήποτε βρίσκεται **αριστερά** του keyword const.

- `char const *buf`

  Μπορώ να αλλάξω τον pointer, αλλά όχι τα δεδομένα του array

- `char *const buf`

  Δεν μπορώ να αλλάξω τον pointer, αλλά μπορώ να αλλάξω τα δεδομένα του array

Διαβάστε [και γράψτε] από δεξιά...

- Ο buf είναι const pointer (άρα δεν αλλάζει) σε χαρακτήρες

  - `char *const buf`

- Ο buf είναι pointer σε const χαρακτήρες (οι οποίοι, συνεπώς δεν αλλάζουν)

  - `char const *buf`

C++ strings

Strings

- Στη C++ υπάρχει έτοιμος τύπος δεδομένων που ονομάζεται string για τη διαχείριση συμβολοσειρών.
- Σε μια μεταβλητή τύπου string μπορούμε να αποθηκεύσουμε ακολουθίες χαρακτήρων χωρίς να ανησυχούμε για το αν χωράνε στην μεταβλητή αυτή.
- Η διαχείριση των bytes που δεσμεύονται από τη μεταβλητή γίνεται δυναμικά ανάλογα με το μέγεθος της ακολουθίας που αποθηκεύουμε σε αυτή.
- Επίσης δεν χρειάζεται να ανησυχούμε για το αν η ακολουθία τερματίζεται με '\0'.

- Χρειάζεται να συμπεριλάβουμε τη βιβλιοθήκη string στην include list του προγράμματος

Strings – Δήλωση και αρχικοποίηση

```
#include <iostream>
#include <string>
using namespace std;
int main(){
 string imBlank;
 string heyMom(“where is my coat?”);
 string heyPap = “whats up?”;
 string heyGranPa(heyPap);
 string heyGranMa = heyMom;
}
```

Strings – Δήλωση και αρχικοποίηση

```
#include <iostream>
#include <string>
using namespace std;
int main(){
 string s1("I saw elvis in a UFO.");
 string s2(s1, 0, 8);

 string s3 = s1 + "Am I crazy?"; //append!!
 string s4 = s1.substr(2, s1.length()-10);
                    //take a substring: starting pos, how many chars
}
```

Strings – Επεξεργασία

```
#include <iostream>
#include <string>
using namespace std;
int main(){
 string s1("I saw elvis in a UFO.");
 cout << s1.size() << "\n";
 cout << s1.length() << "\n";   // Very important, very frequently used!
 cout << s1.capacity() << "\n";
 string s2 = " thought I ";
 s1.insert(1, s2);              // insert in position 1, i.e. right after 'I'
 cout << s1.capacity() << "\n";
 string s3 = "I've been working too hard";
 s1.append(s3);                 //append, again, equiv. to +
 s1.append(", or am I crazy?");
 s1.resize(10);                 // increase/reduce the capacity
 cout << s1 << "\n";
}
```

Strings – Επεξεργασία

```
#include <iostream>
#include <string>
using namespace std;
int main(){
 string s(“Hello mother”);
 string s1(“fa”);
 s.replace(6, 2, s1);
        // start at 7th character (counting starts with 0), replace 2 chars with the whole of s1
        // Syntax: starting  pos, how many chars to replace, replacement
 cout << s << “\n”;
}
```

Strings – Επεξεργασία

```
#include <iostream>
#include <string>
using namespace std;
int main(){
  string s("Hello mother. How are you mother? Im fine mother.");
  string s1("fa");
  string s3("mo");
  int start = 0;
  unsigned int found = 1;

  found = s.find(s3, start);
  while (found != string::npos){
    s.replace(found, s3.length(), s1);
    start = found + s1.length();
    cout << s << "\n";
    found = s.find(s3, start);
  }
  cout << s << "\n";
}
```

Strings – Επεξεργασία

```
s.rfind(s1, s.length()-1);
```
// finds s1 in s backwards starting from the last character

```
s.find_first_of("@$.", pos);
```
// find position of first occurrence of a char in "…" starting from pos

```
s.find_last_of("@$.", pos);
```
// find position of last occurrence of a char in "…"

Strings – Επεξεργασία με τελεστές

```
s= s1 + s2 + s3;
s += s5 + s6;
s[10] = 'c';
s.at(10) = 'c';
s1 == s2
s1 != s2
s1 >= s2
s1 <= s2
s1 > s2
s1 < s2
```

Strings and backwards compatibility with C

- Για λόγους συμβατότητας προς τα πίσω, υποστηρίζονται και οι συμβολοσειρές της C
- Στη C τα strings είναι ακολουθίες χαρακτήρων που τερματίζονται με το χαρακτήρα '\0'.
- Αποθηκεύονται σε πίνακες χαρακτήρων.
- Η επεξεργασία τους διευκολύνεται από έτοιμες συναρτήσεις όπως strcpy, strcmp,…

- How to convert a string `s` to a (C-style) char array:

```
const char *p = s.c_str();
```

(συνθήκες, βρόγχοι και τα σχετικά)

ΒΑΣΙΚΕς ΕΝΤΟΛΕς

If … Else … Statement

```
if (condition) {
    statement;
    ...
}
else {
 statement;
    ...
}
```

Λογικοί τελεστές για σύνθετες συνθήκες:

OR: `((cond1)||(cond2))`

AND: `((cond1)&&(cond2))`

NOT: `!(cond1)`

Προσοχή: **`= vs. ==`**

Switch Statement

```
switch (condition) {
   case constant1:
        statement;
        ...
        break;
   case constant2:
        statement;
        ...
        break;
   ...
   default:
        statement;
        ...
        break;
}
```

Προσοχή: αν παραληφθεί το **`break`**, εκτελείται η επόμενη εντολή!

Δεν το παραλείπουμε ποτέ!!!

Το `default` καλύπτει την περίπτωση που χάνουμε κάποια case

Loop Statements

```
while (condition){
   statement;
   ...
}

for (declaration-initialization; condition; iteration){
   statement;
   ...
}

//Example of counting...
for (int i=0; i<5; i++){          //runs 5 times
   cout << i*5+3;
}
```

Συναρτήσεις (εισαγωγικά)

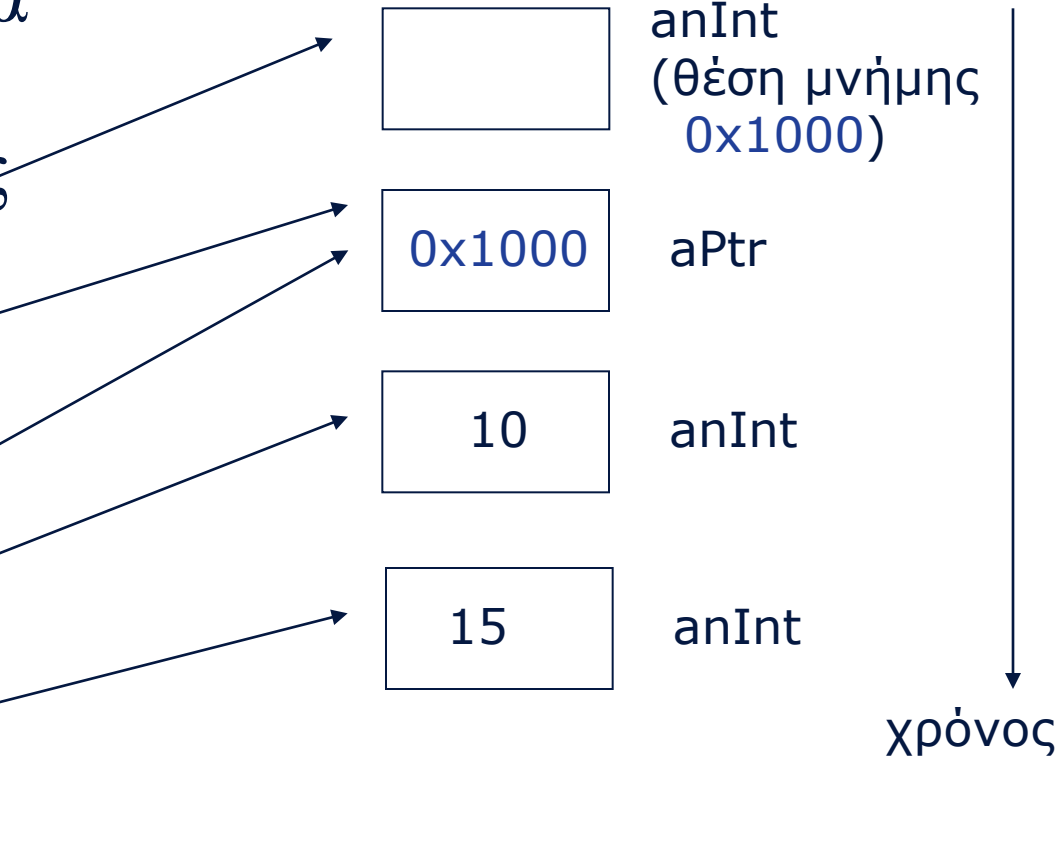
```
#include <iostream>
using namespace std;

float triangleArea (float width, float height);
main(){
        float area = triangleArea(1.0, 2.0);
        cout << "The area of the triangle is "  <<
   area;
}
float triangleArea(float width, float height){
         float area;  // local στην triangleArea
         area = width * height / 2.0;
         return (area);
}
```

ΔΕΙΚΤΕΣ ΚΑΙ ΑΝΑΦΟΡΕΣ

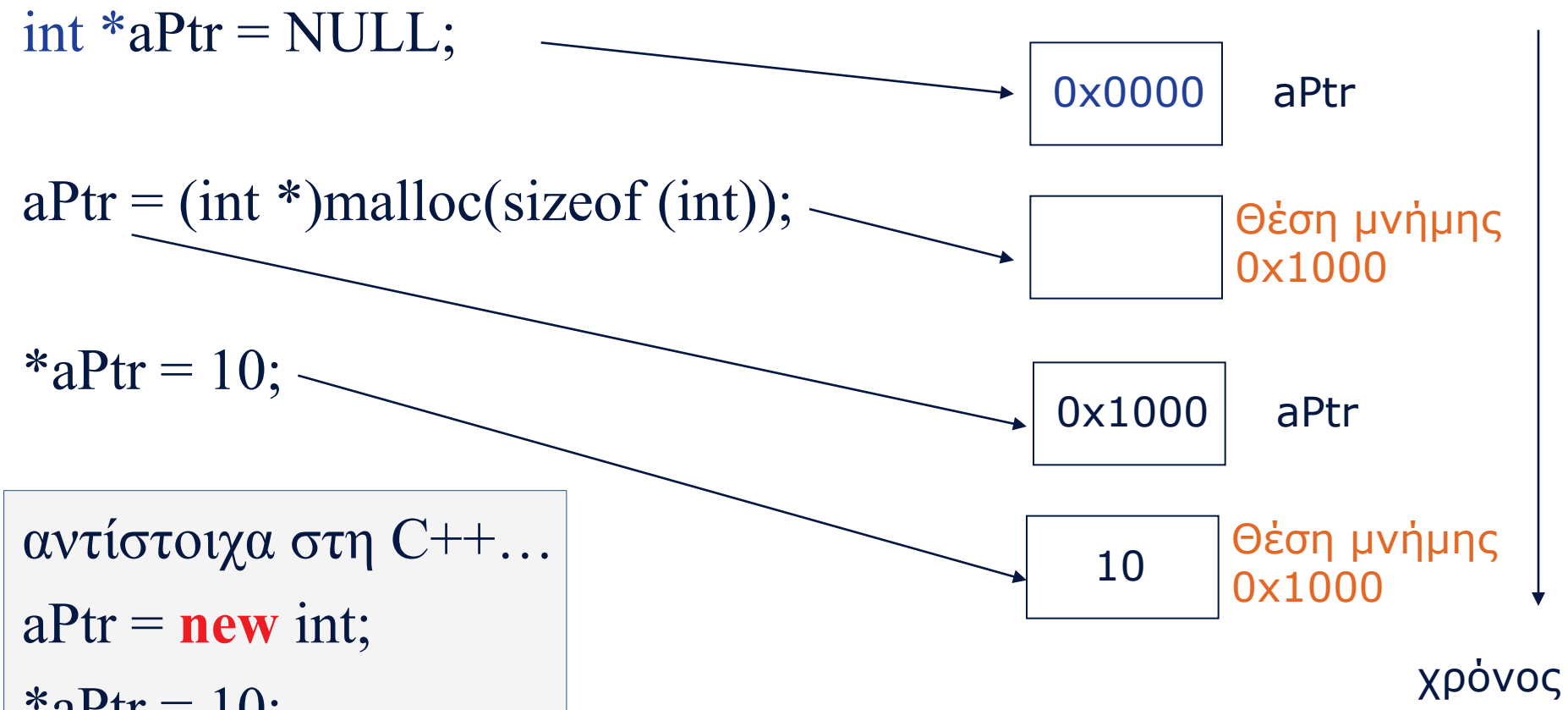
Δείκτες

- Δείκτης είναι μια μεταβλητή στην οποία αποθηκεύουμε τη διεύθυνση μιας άλλης μεταβλητής.

```
int anInt;
int *aPtr;

aPtr = & anInt;
anInt = 10;
*aPtr = 15;
```



Δείκτες – Δέσμευση Μνήμης



αντίστοιχα στη C++…

aPtr = **new** int;

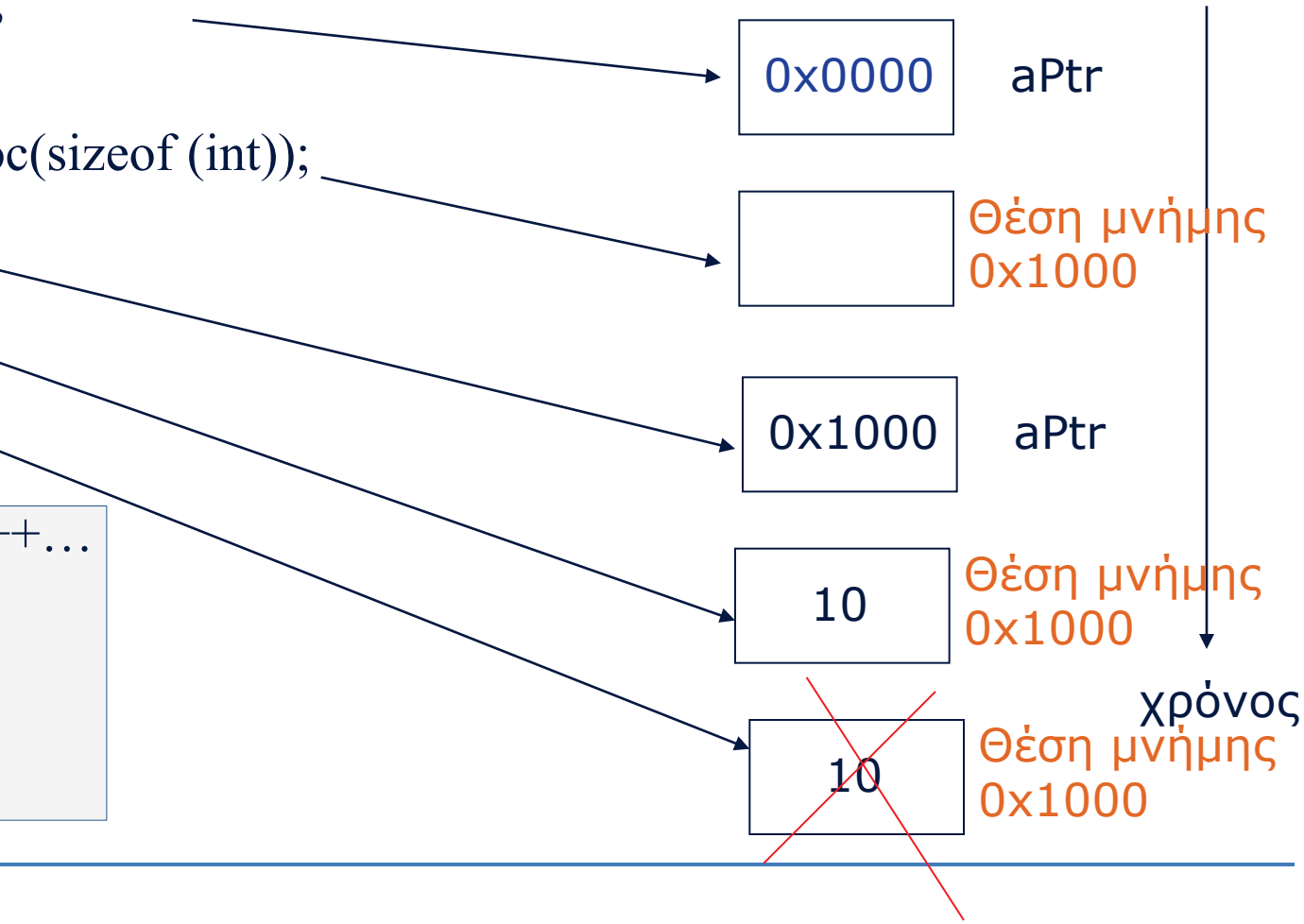
*aPtr = 10;

Δείκτες – Αποδέσμευση Μνήμης

```
int *aPtr = NULL;

aPtr = (int *)malloc(sizeof (int));

*aPtr = 10;
free(aPtr);
```

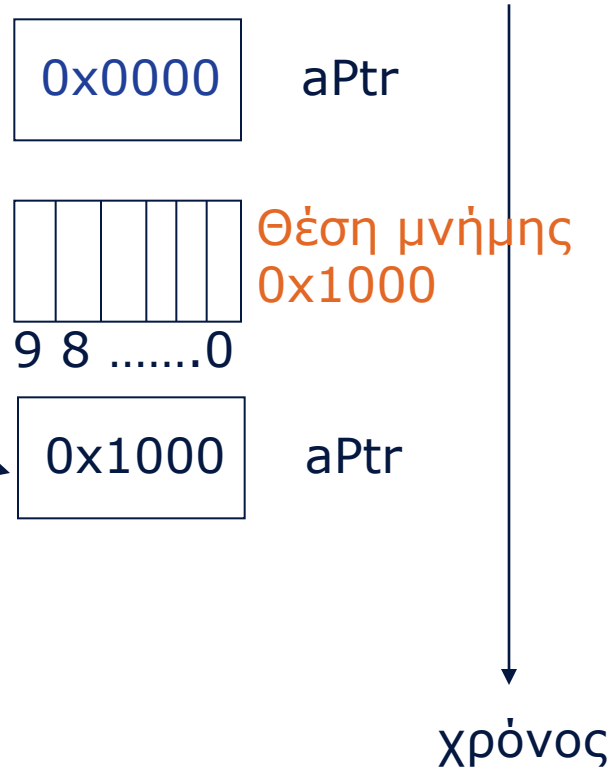
αντίστοιχα στη C++…

```
aPtr = new int;
*aPtr = 10;
delete aPtr;
```



Δείκτες – Πίνακες - Δέσμευση Μνήμης

```
int *aPtr = NULL;

aPtr = (int *)malloc(10*sizeof (int));

*aPtr = 10; ⇔ aPtr[0] = 10;
*(aPtr + 2) = 6; ⇔ aPtr[2] = 6;
free(aPtr);
```



αντίστοιχα στη C++…

```
aPtr = new int [10];
*aPtr = 10;
delete [] aPtr;
```

Αναφορές (References)

- Μια ***αναφορά*** σε μια μεταβλητή είναι σαν ένα συνώνυμο για τη μεταβλητή.
- **Προσοχή**: ΔΕΝ πρόκειται για pointers (αν και υπάρχουν κάποιες ομοιότητες, βέβαια – βλ. παρακάτω...)

```
...
int myInt;
int &myRef = myInt;
...
```

Έτσι δηλώνουμε αναφορές:

κατά τη δήλωση πρέπει να καθορίζουμε τη μεταβλητή της οποίας είναι συνώνυμα

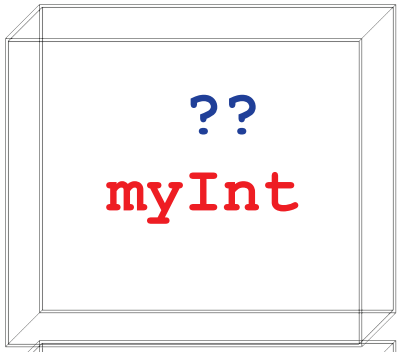
Αναφορές (References)

- Όταν δηλώνουμε μια μεταβλητή, αυτό σημαίνει ότι της παραχωρούμε κάποιο χώρο στη μνήμη και ένα όνομα στο χώρο αυτό
- Όταν δηλώνουμε μια αναφορά σε μια μεταβλητή, είναι σαν να δίνουμε παραπάνω από ένα ονόματα στο συγκεκριμένο χώρο στη μνήμη.
- Οι παρακάτω εντολές είναι ισοδύναμες:

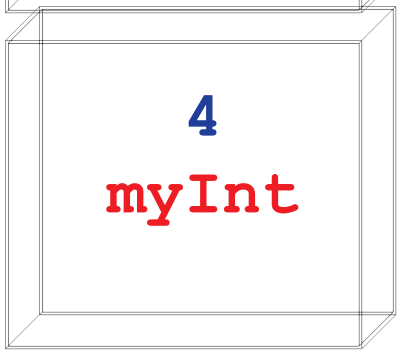
```
myInt++;
myRef++;
```

Αναφορές

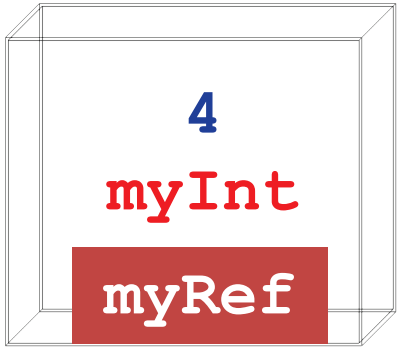
```
int myInt;
```




```
myInt = 4;
```




```
int &myRef =
myInt;
```



Τι κρύβουν ....

```
int myInt = 5;
int &myRef = myInt;
```

```
int * const myRef_p = &myInt;
```

```
myInt = 8;
```

```
myInt = 8;
```

```
myRef = 7;
```

```
*myRef_p = 7;
```

Τι κρύβουν ....

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

int main() {
        int a(1), b(3);
        int& refA = a;

        cout << refA << endl;
        cout << a << endl;

        refA = b;
        cout << a << endl;
        return 0;
}
```

```
int * const refA_p = &a;

cout << *refA_p << endl;

*refA_p = b;
```

Πού χρησιμεύουν ….

- Οι δηλώσεις αναφορών που παίζουν το ρόλο συνώνυμων μιας μεταβλητής σε ένα πρόγραμμα δεν έχουν πολύ μεγάλη χρησιμότητα . Το ίδιο ισχύει για τις δηλώσεις δεικτών που τοποθετούνται σε κάποια υπάρχουσα μεταβλητή

```
int x = 10;
int *x_p = &x;
```

- Εκεί που βοηθούν δραστικά τόσο οι δείκτες όσο και οι αναφορές είναι σε συναρτήσεις στο πέρασμα παραμέτρων δι' αναφοράς.
- **Στην περίπτωση που χρησιμοποιήσουμε αναφορές σαν ορίσματα σε συναρτήσεις το πέρασμα δι' αναφοράς γίνεται ακόμα πιο απλό.**

ΠΈΡΑΣΜΑ ΠΑΡΑΜΈΤΡΩΝ ΣΕ ΣΥΝΑΡΤΉΣΕΙς

Πέρασμα δια τιμής

```
#include <iostream>
Using namespace std;
void increase(int myParameter)
{
    myParameter++;
    cout << myParameter << "\n";
}
int main(){
    int aValue = 3;

    increase(aValue);
    cout << aValue << "\n";
}
```

Δημιουργείται τοπική κόπια του `aValue`

Πέρασμα δι'αναφοράς με δείκτη

```
#include <iostream>
using namespace std;
void increase(int *myPointer)
{
    (*myPointer)++;
    cout << *myPointer << "\n";
}
int main(){
    int aValue = 3;

    increase(&aValue);
    cout << aValue << "\n";
}
```

Πέρασμα δι'αναφοράς με αναφορά

```
#include <iostream>
using namespace std;
void increase(int &myRef)
{
    myRef++;
    cout << myRef << "\n";
}
int main(){
    int aValue = 3;

    increase(aValue);
    cout << aValue << "\n";
}
```

Πέρασμα δι' αναφοράς με αναφορά – τι κρύβει…

```
#include <iostream>
using namespace std;
void increase(int &myRef)
{
    myRef++;
    cout << myRef << "\n";
}
int main(){
    int aValue = 3;

    increase(aValue);
    cout << aValue << "\n";
}
```

```
Increase(int *const myRef_p)
(*myRef_p)++;
cout << *myRef_p << "\n";
localIncrease(&aValue)
```

Παράδειγμα – μεταβλητές που δεν αλλάζουν

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

void showConst(int &counter, const int &newCounter){
        counter = counter + newCounter;
    /* will not pass!!!  newCounter++; */
}
int main(){
   int aCounter = 0;
   int anotherCounter = 3;
   showConst(aCounter, anotherCounter);
   cout << aCounter << "\n";
}
```

Arrays as parameters

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

void printArray(int arg[], int length){
  int i;
  for (i=0;i<length;i++){
      cout << arg[i] << "\n";
  }
}
main(){
  int firstArray[] = {5,10,15};
  int secondArray[] = {3,6,9,12};
  printArray(firstArray,3);
  printArray(secondArray,4);
}
```

Διαφορές pointers και references

- Μια αναφορά δεν στέκει ποτέ μόνη της σε ένα πρόγραμμα. Πρέπει να δηλωθεί οπωσδήποτε σαν αναφορά σε κάποια μεταβλητή (απόρροια του γεγονότος ότι οι const μεταβλητές θέλουν αρχικοποίηση).
- Ένας pointer μπορεί κάλλιστα να μη δείχνει πουθενά.
- Αν μια αναφορά δείχνει σε μια μεταβλητή, ΔΕΝ μπορώ να την βάλω να δείξει σε άλλη μεταβλητή (απόρροια του γεγονότος ότι οι const μεταβλητές δεν αλλάζουν).
- Έναν pointer μπορώ να τον μετακινώ κατά βούληση.

Ομοιότητες pointers και references

- Μπορούν να περνούν αμφότερα σαν παράμετροι σε μια συνάρτηση.
- Ομοίως, μπορούν να αποτελούν την τιμή επιστροφής μιας συνάρτησης.

Ισοδύναμο αποτέλεσμα

```
void byPointer(int *value){
   *value +=5;
}

main(){
   int i = 3;
   byPointer(&i);
}
```

```
void byRef(int &value){
   value +=5;
}

main(){
   int i = 3;
   byPointer(i);
}
```

Συνοπτικά για το πέρασμα παραμέτρων

`function (int var)` Πέρασμα τιμής	Η μεταβλητή περνιέται αυτούσια ως παράμετρος στη function, μπορεί να αλλάξει τοπικά στη function, αλλά οι αλλαγές δεν περνούν εξωτερικά στο πρόγραμμα.
`function (const int var)` Πέρασμα τιμής	Όπως πριν, αλλά χωρίς να μπορεί να αλλάξει η τιμή της μεταβλητής εσωτερικά της function
`function (int &var)` Αναφορά	Περνιέται μια αναφορά ως παράμετρος. Ότι αλλαγές γίνουν στην αναφορά, ανακλώνται και εξωτερικά στο πρόγραμμα
`function (const int &var)` **Σταθερή Αναφορά**	Όπως πριν, αλλά χωρίς να μπορεί να αλλάξει η τιμή της αναφοράς εσωτερικά της function
`function (int array [])`	Η C++ αυτομάτως μετατρέπει τα arrays σε αναφορές
`function (int *var)`	Περνά ένας pointer ως παράμετρος της function

Συνοπτικά για το πέρασμα παραμέτρων

Excellent guideline at Google Code Style:

- **use const references for input variables that are not to be changed**
- **use pointers for output variables that will be changed, eventually**

Significantly enhances **maintainability** and usage: the reader knows what is input + what is output
If the maintainer accidentally tries to mess with the (const) input, compiler catches it

```
void updateLicense(const int &newNumber, int *licenceNo)
```

Default τιμές στο πέρασμα παραμέτρων

```
void f (int x = 1);
void g (int a, int b = 0){
   cout << "a: " << a << " b: " << b << "\n";
}

int main ()
{
    f(5);
    f();
    g(1, 2);
    g(5);
}

void f (int x){ … //could be (int x =1 )
}
```

Default τιμές στο πέρασμα παραμέτρων

```
void h(int a, int b = 0, int c, int d = 0){
   // ……..
}

int main (){
   h(10, 5, 20);
}
```

Avoid default values if you can: source of much confusion!!

- στην περίπτωση αυτή σίγουρα a = 10
  - το 5 μπορεί να αντιστοιχεί στο b οπότε c = 20 και d = 0
  - Η' το b = 0, c = 5 και d = 20

- προς αποφυγή του παραπάνω προβλήματος το οποίο δεν μπορεί να επιλύσει ο compiler **αν η i-οστή παράμετρος στη δήλωση μιας συνάρτησης με N παραμέτρους έχει default τιμή, όλες όσες την ακολουθούν πρέπει επίσης να έχουν δηλωμένες default τιμές**.

με βάση το παραπάνω αν κατά την κλήση η συνάρτηση έχει ι <= κ < N πραγματικές παραμέτρους, οι N-κ τελευταίες λαμβάνουν τις default τιμές

Κονσόλα -- Αρχεία

C++ I/O

C++ IO Streams

IOSTREAMs - reading lines

```
#include <iostream>
using namespace std;
int main(){
   char buf[100]; char whatIsLeft[200];

/* reads max of 9 chars or up to '\n' and leaves the '\n' behind, i.e. the next char we read is \n */
   cout << "Give a string, at most 9 chars, ending with endl \n";
   cin.get(buf, 10, '\n');
   cout << buf << endl;

   cout << "Here, normally, you should give another string \n";
   cin.get(whatIsLeft, 10, '\n');
   cout << whatIsLeft << endl;

}
//Try declaring buf as string (including #include <string>) to see what happens
```

IOSTREAMs - reading lines

```
#include <iostream>
using namespace std;
int main(){
   char buf[100]; char whatIsLeft[200];

/* reads max of 9 chars or up to '\n' including the '\n' which is not included in buf */
   cout << "Give a string of at most 9 characters, ending with
   endl \n";
   cin.getline(buf, 10, '\n');
   cout << buf << endl;

   cout << "Here, normally, you should give another string \n";
   cin.getline(whatIsLeft, 10, '\n');
   cout << whatIsLeft << endl;

}

// Try it with input that exceeds the limit of 10 chars and with input less than 10 chars
```

IOSTREAMs - reading lines

```
#include <iostream>
using namespace std;
int main(){
   char buf[100];

   while (cin.eof() != true){ // or cin.good() == true
        cout << "Give another string: ";
        cin.getline(buf, 100, '\n');
        cout << buf << endl;
   }
}

//Eof is given via Ctrl+D
```

IOSTREAMs

```
#include <iostream>
using namespace std;
int main(){
   float b; int a;

   while (cin.good() == true){ // or cin.eof() != true
        cout << "Give a float and an int: ";
        cin >> b >> a;
        cout << b <<"\t" << a << endl;
   }
}
```

//Try it separating the 2 numbers with space, enter, or tab

IOSTREAMs - reading lines σε string

```
#include <iostream>
#include <string>
using namespace std;
int main(){
  string s;

  while (getline(cin, s)){
    cout << s << endl;
  }
}
```

Very useful!

Can handle strings of
(i) arbitrary size and
(ii) several whitespaces

Contrast this to
`cin >> s`
that can withstand only one (1) word in the input stream

//Try it with strings having several whitespaces and big size

C++ File Streams

File streams

- Για ανάγνωση από αρχείο δημιουργούμε μεταβλητές (ρεύματα) τύπου ifstream
- Για γράψιμο σε αρχείο δημιουργούμε μεταβλητές (ρεύματα) τύπου ofstream
- Ανάγνωση γίνεται με >> για βασικούς τύπους (int, float, char, char[], string…) ΚΑΙ get, getline για char[]
- εγγραφή γίνεται με << για βασικούς τύπους (int, float, char, char[], string,…)
- Γενικά ότι συνάρτηση είδαμε στα cin, cout ισχύει και σε μεταβλητές ifstream, ofstream.

Copy one file to another, line by line

```
#include <iostream>
#include <cstdlib>
#include <fstream>
using namespace std;
int main(){
  char buf[100];

  ifstream in("in.txt");
  ofstream out("out.txt");
  if(in.good()!= true) exit(EXIT_FAILURE);

  while (in.eof() != true){
    in.getline(buf, 100, '\n');
    out << buf << endl;
  }
  in.close();
  out.close();
}
```

Same old story:

Μια μεταβλητή για κάθε αρχείο

ATTN: αν κάτι πάει στραβά, πιάστε το νωρίς!!

Διαβάζουμε γραμμή – γραμμή μέχρι τέλους του αρχείου && … we do sth with it … (here: copy it to out.txt)

housekeeping

Append a file to another file, line by line

```
#include <iostream>
#include <cstdlib>
#include <fstream>
using namespace std;
int main(){
  char buf[100];
  char c;

  ifstream in("in.txt");
  ofstream out("out.txt", ios::app);   // ios::out is the default
  if(in.good()!= true) exit(1);
  while (in.eof() != true){
    in.getline(buf, 100, '\n');
    out << buf << endl;
  }
  in.close();
  out.close();
}
```

Append a file to another file, line by line

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
int main(){
  char buf[100];

  ifstream in;
  ofstream out;
  in.open("in.txt");
  out.open("out.txt", ios::app); // ios::out default
  if(in.good()!= true) exit(1);
  while (in.eof() != true){
    in.getline(buf, 100, '\n');
    out << buf << endl;
  }
  in.close();
  out.close();
}
```

Η διαφορά με το προηγούμενο παράδειγμα είναι στη χρήση της open

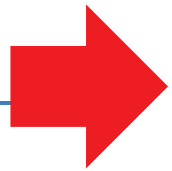
Extra methods

```
  in.clear();                    // sets eof flag back to 0
  in.seekg(0,ios::beg);          // offset from the beginning
  while (in.eof() != true){
    in.getline(buf, 100, '\n');
    cout << buf << endl;
  }

  out.seekp(0,ios::beg);         // offset from the beginning
  out << "BBBBBBBBBBB";
  in.close();
  out.close();
}
```

// seekp seekg χρήση και με ios::end και αρνητικό offset, ή ios::curr

Try it for yourselves (part 1)

```
#include <iostream>
#include <cstdlib>
#include <fstream>
using namespace std;
int main(){
  char buf[100];
  char c; int i; float f;

  fstream in;
  fstream out; // ios::out is default
  in.open("in.txt", ios::in);
  out.open("out.txt", ios::out|ios::in); // ios::app to append
  if(out.good()!= true) exit(1);
  while (in.eof() != true){
    in.getline(buf, 100, '\n');
    out << buf << endl;
  }
```



…continued: part 2

```
  in.clear();
  in.seekg(0,ios::beg);
  while (in.eof() != true){
    in.getline(buf, 100, '\n');    //print file on screen
    cout << buf << endl;
  }

  out.seekg(0,ios::beg);
  while (out.eof() != true){
    out.getline(buf, 100, '\n');
    cout << "XXX" << buf << endl; //print on screen file's info with XXX in front…
  }
  out.clear();
  out.seekp(0,ios::beg);                //replace file's starting bytes with BBBB…
  out << "BBBBBBBBBBB";
  in.close();
  out.close();
}
```

(structs, υπερφόρτωση συναρτήσεων, namespaces, …)

POT POURRI

Disclaimer

- Δεν γίνεται μονομιάς να μάθετε ολόκληρη τη C++
  - Υπάρχουν θέματα που θα τα ανακαλύψετε σιγά σιγά, έξω από το πλαίσιο του μαθήματος
  - Υπάρχουν και στοιχεία που θα πρέπει να αντιμετωπίσετε με πολύ προσοχή – ενδεχομένως και να αποφασίσετε να μην τα χρησιμοποιήσετε καθόλου…
- Στη συνέχεια δίνουμε μια πρώτη εικόνα από στοιχεία της C++ στα οποία δε θα επεκταθούμε ιδιαίτερα…
- Θυμηθείτε ότι ο σκοπός του μαθήματος είναι να διδάξουμε βασικές αρχές αντικειμενοστρεφούς προγραμματισμού και όχι C++

Structs & Κλάσεις

Structs

- Ο πιο απλός τρόπος δήλωσης ενός struct είναι ως εξής:

```
struct structName {
   fieldType fieldName;
   fieldType fieldName;
   ...
};
```

Τα structs είναι τύποι δεδομένων

- Δήλωση struct

```
struct complex {  //μιγαδικός
    double re;
    double im;
};
```

- Δήλωση μεταβλητής τύπου complex

```
complex z;
```

  - Σε **αντίθεση με τη C** δεν χρειάζεται να γράψουμε

```
struct complex z;
```

Συναρτήσεις μέσα σε structs

- Σε αντίθεση με τη C μπορούμε να ορίσουμε μεθόδους για τα structs

```
struct complex {      //μιγαδικός
  double re;
  double im;
  void show();
};

void complex::show(){
  cout << "(" << re << "," << im << \ ")\n";
}
```

Συναρτήσεις μέσα σε structs

- Εντελώς αντίστοιχα μπορώ να ορίσω την κλάση

```
class complex {        //μιγαδικός
public:
  double re;
  double im;
  void show();
};

void complex::show(){
  cout << "(" << re << "," << im << \ ")\n";
}
```

Δηλώσεις και χρήση αντικειμένων

```
int main ()
{
  complex a;

  a.re = 1.0;
  a.im = 2.0;
  a.show();

  return 0;
}
```

Εναλλακτικά (α-λα-C):

```
#include <iostream.h>
...
void show (complex &p){
        ...
}

main(){
        complex a;
        ...
        show(a);
}
```

Δείκτες σε αντικείμενα

```
int main ()
{
  complex *pa, *pb;
  complex a;

  pa = &a;
  pb = new complex;
  a.re = 1.0; pa ->re = 1.0; (*pa).re = 1.0;
  a.im = 2.0; pa->im = 2.0;  (*pa).im = 2.0;
  a.show();   pa->show();    (*pa).show();

  pb->re = 3.0; pb->im = 4.5;
  pb->show();

  delete pb;
  return 0;
}
```

Υπερφόρτωση (Overloading) Συναρτήσεων

Υπερφόρτωση συναρτήσεων

- Στη C++ επιτρέπεται να δηλώσουμε συναρτήσεις με το ίδιο όνομα αλλά:
  - με διαφορετικό αριθμό παραμέτρων
  - με διαφορετικούς τύπους παραμέτρων
  - **ΠΡΟΣΟΧΗ!** Όχι συναρτήσεις που διαφέρουν μόνο στον τύπο του αποτελέσματος που επιστρέφουν.

Υπερφόρτωση συναρτήσεων

```
#include <iostream>

using namespace std;

int max (int a, int b){
  if (a > b) return a;
  else return b;
}

int max (int a[], int size){
  int max = a[0];

  for (int i = 0; i < size; i++)
    if (a[i] > max) max = a[i];

  return max;
}
```

Υπερφόρτωση συναρτήσεων

```
int main(){
  int A[] = {-2, 4, 5, 6};
  int a = 3, b = 5;

  int ret;

  ret = max(A, 4);
  cout << ret << endl;

  ret = max(a, b);
  cout << ret << endl;
}
```

Υπερφόρτωση συναρτήσεων

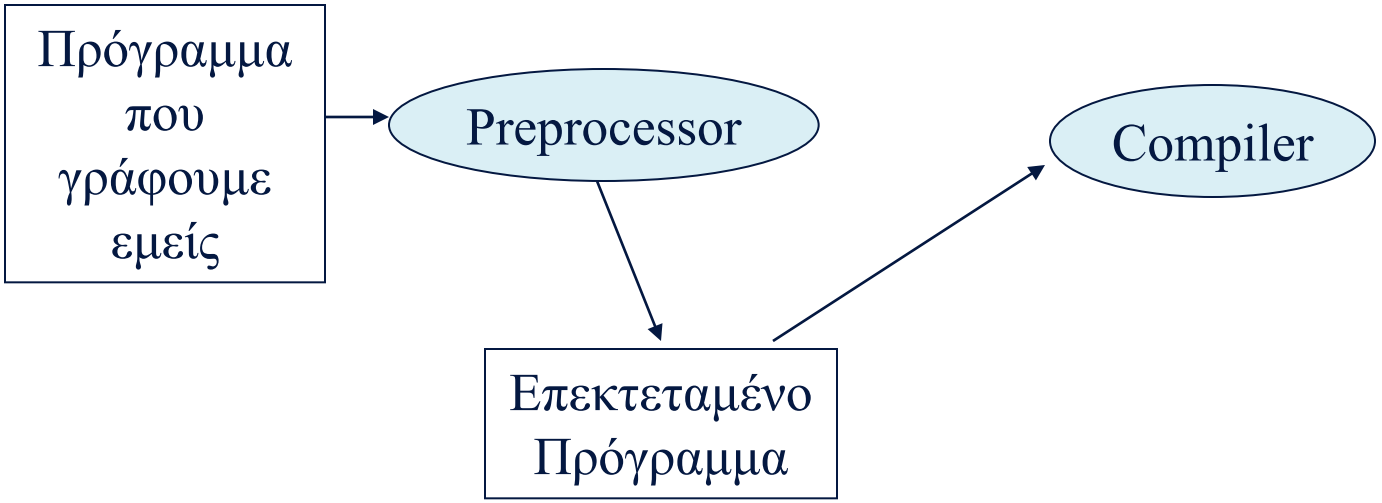
```
#include <iostream>

using namespace std;

int max (int a, int b){
  if (a > b) return a;
  else return b;
}

float max (int a, int b){
   if (a > b) return a;
   else return (float) b;
}
```

Υπερφόρτωση συναρτήσεων

```
int main(){
  int a = 3, b = 5;

  int ret;

  ret = max(a, b);
  cout << ret << endl;

  max(a, b); // Πρόβλημα ποια θα καλεστεί ???
             // Compile Error
}
```

C++ Preprocessor

Preprocessor της C++ (#include, #define, …)



#include

- Ενσωματώνει στο πρόγραμμά μας αυτούσια αρχεία.
- `#include <iostream.h>`
- `#include “mydefinitions.h”`
- `#include “../../mydefinitions.h”`

  DOS/Windows: `“..\..\mydefinitions.h”`

#define

- `#define SIZE 20`
- Σημαίνει ότι οπουδήποτε βλέπει ο preprocessor `SIZE`, το αντικαθιστά με `20`.

```
//illegal definitions
#define X = 5
#define X 5;
```

- Σαφώς καλύτερα να χρησιμοποιεί κανείς `const`

```
const int SIZE = 20;
```

Inline Functions

Inline functions

Είναι συναρτήσεις οι οποίες χρησιμοποιούνται από τον compiler στην παραγωγή του εκτελέσιμου κώδικα: για επιτάχυνση της διαδικασίας, ενσωματώνονται στον κώδικα της καλούσας συνάρτησης, αντί να κληθούν ως χωριστές συναρτήσεις.

Χρήσιμες ΜΟΝΟ για πολύ μικρές συναρτήσεις.

```
inline int square (int value){
        return (value * value);
}
```

Χρησιμοποιούνται κανονικά στο πρόγραμμα, π.χ.,

```
main(){
        …
        mySquareArea = square(squareEdge);
        …
}
```

Namespaces

Namespaces

- Πολλές φορές στη C φτιάχνουμε συναρτήσεις των οποίων τα ονόματα έρχονται σε σύγκρουση με έτοιμες συναρτήσεις της γλώσσας
  - σαν αποτέλεσμα έχουμε λάθη μετάφρασης που δεν εξηγούνται εύκολα…
  - και κόστος επιδιόρθωσης του κώδικα
- Ένα πρόγραμμα μπορεί να συνθέτει κώδικα από δύο ή περισσότερα άτομα που μπορεί να χρησιμοποιούν συνώνυμες συναρτήσεις ή κλάσεις.
  - σαν αποτέλεσμα πάλι μπορεί να έχουμε λάθη μετάφρασης του συνολικού κώδικα
- στη C++ υπάρχει ο μηχανισμός namespace που μπορεί να χρησιμοποιηθεί για την αποφυγή συγκρούσεων

```
namespace X {
……..
};
```

Παράδειγμα

- **Στο αρχείο: `util.h` υπάρχουν χρήσιμες συναρτήσεις που πήραμε από κάποιον άλλο προγραμματιστή**

…….

```
float squareArea(float length){
        return (length*length);
}
```

…….

```
int f(){….}
```

Παράδειγμα (συνέχεια)

```
#include <iostream.h>
#include "util.h"

//1 square mile is 2.59 square km
//the following function returns square miles
//whereas length is in km

float squareArea(float length){
        return (length*length / 2.59);
}

main (){
  cout << "Normal " << squareArea(2.0);
   f();
}
```

Παράδειγμα

- **Στο αρχείο: `util.h`**

```
namespace UtilJohn {

    float squareArea(float length){
        return length*length;
     }
     int f(){….}

}
```

Παράδειγμα (συνέχεια)

```
#include <iostream.h>
#include "util.h"

namespace Local{
                    //1 square mile is 2.59 square km
                    //the following function returns square miles
                    //whereas length is in km
    float squareArea(float length){
        return (length*length / 2.59);
    }
}
main (){
   cout << "Normal " << Local::squareArea(2.0);
   UtilJohn::f();
}
```

Παράδειγμα (συνέχεια)

```
#include <iostream.h>
#include "util.h"

namespace Local{        //returns square km for length in km
                        //1 square mile is 2.59 square km
                        //the following function returns square miles
                        //whereas length is in km
    float squareArea(float length){
        return (length*length / 2.59);
    }
}
using namespace Local;
main (){
   cout << "Normal " << squareArea(2.0);
   UtilJohn::f();
}
```

Παράδειγμα (συνέχεια)

```
#include <iostream.h>
#include "util.h"

namespace Local{      //returns square km for length in km
                      //1 square mile is 2.59 square km
                      //the following function returns square miles
                      //whereas length is in km
    float squareArea(float length){
        return (length*length / 2.59);
    }
}
using namespace Local;
main (){
   cout <<"English "<< UtilJohn::squareArea(2.0)
                << "\n";
   cout << "Normal " << squareArea(2.0);
}
```

Το `std` namespace

- Μπορείτε να δηλώνετε το **`std`** namespace:

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

- για να μη χρειάζεται να το γράφετε συνέχεια

πχ `std::cout <<` ….

Αν και κάποιοι μεταφραστές το κάνουν αυτόματα…

Exceptions

Εξαιρέσεις

- Αρκετές φορές στην πράξη υλοποιούμε συναρτήσεις που δέχονται σαν όρισμα κάποια δεδομένα και επιστρέφουν ένα αποτέλεσμα κάποιου τύπου (π.χ. int).
- Το αποτέλεσμα σε πολλές περιπτώσεις μπορεί να είναι οποιαδήποτε τιμή αυτού του τύπου (int).
- Παρόλα αυτά αν κάτι πάει στραβά στην εκτέλεση της συνάρτησης θέλουμε να επιστρέφει κάτι σε αυτόν που την κάλεσε ώστε να καταλάβει ότι κάτι πήγε στραβά (π.χ. απέτυχε το άνοιγμα ενός αρχείου).
- Για αυτό το κάτι, προφανώς δεν μπορούμε να δεσμεύσουμε μια τιμή αν αυτή μπορεί να επιστρέφεται και κάτω από κανονικές συνθήκες (π.χ. -1).
- Αντί αυτού μπορούμε να χρησιμοποιήσουν το μηχανισμό των εξαιρέσεων…..

Εξαιρέσεις

- Γενικά μπορούμε να πούμε ότι μια εξαίρεση είναι μια δομή (struct) που επιστρέφει μια συνάρτηση όταν κάτι πάει λάθος.
- Η επιστροφή γίνεται με throw αντί για return.
- Η συλλογή αυτής της δομής - εάν και εφόσον επιστραφεί – γίνεται με τη χρήση της εντολής try/catch.

Εξαιρέσεις

```
#include <iostream>
#include <string>
using namespace std;
struct MyError {
  int code;
  string cause;
};
// -1 is a valid calculation of the function
float test_function(string s) {
  MyError e;
  ifstream in;

  in.open(s);
  if (in.good() != true) {
        e.code = 555;
        e.cause = “open file failed";
        throw e;
  }
  else {
   float sum;
    // ……. read float numbers from file and return the sum
   return sum;
  }
}
```

Εξαιρέσεις

```
.........................


main(){
 int y; float k;
 try {

  k = test_function ("lala.txt");     // test_function is invoked
                                      // k = ... if ok
 } catch (MyError error) {            // error = ... if a problem occurs
   cout << error.code << error.cause << endl;
 }
```

Εξαιρέσεις - new

```
.........................


main(){
 int * x;

 try {
  x = new int [1000];
 } catch (bad_alloc bad){
   cout << "Bad allocation integer array";
 }
}
```

Υπερφόρτωση Τελεστών

Υπερφόρτωση τελεστών

- Στη C οι τελεστές (+, -, *, ==, >, <, ….) ορίζονται και μπορεί να χρησιμοποιηθούν μόνο μεταξύ μεταβλητών ή σταθερών κάποιου βασικού τύπου (int, float, double, char,……).
- Στη C++ μπορούμε να επανα-ορίσουμε τη λειτουργία των τελεστών για περιπτώσεις μεταβλητών των οποίων ο τύπος ορίστηκε από τον προγραμματιστή.
  - μεταβλητές τύπου struct
  - αντικείμενα κάποιας κλάσης
    - περισσότερα θα πούμε στην περίπτωση αντικειμένων…..

Υπερφόρτωση τελεστών

```
struct complex {
   double re, im;
};

complex operator + (complex x, complex y)
{
   complex result;
   result.re = x.re + y.re;
   result.im = x.im + y.im;
   return result;
}

int operator == (complex x, complex y)
{
   return (x.re == y.re)
       && (x.im == y.im);
}
```

Υπερφόρτωση τελεστών

```
int main ()
{
   complex a, b, c;

   a.re = 1.0; a.im = 2.0;
   b.re = 4.0; b.im = 1.0;

   if (a == b)
      cout << "they are equal\n";
   else
      cout << "they are different\n";
   c = a + b;

   return 0;
}
```