ΣΤΟΙΧΕΙΑ ΤΗΣ ΓΛΩΣΣΑΣ C++

Και ομοιότητες και διαφορές με την C

ΑΝΑΚΕΦΑΛΑΙΩΣΗ

Άσκηση

- Θέλουμε να φτιάξουμε ένα «παιχνίδι» το οποίο κάνει το εξής:
  - Έχουμε δυο αυτοκίνητα που ξεκινάνε από το σημείο 0 της ευθείας και κινούνται τυχαία πάνω στις τιμές των ακεραίων.
    - Σε κάθε κίνηση διαλέγουν τυχαία να πάνε αριστερά, δεξιά, ή να μείνουν στο ίδιο σημείο.
  - Το παιχνίδι σταματάει όταν τα δυο αυτοκίνητα συγκρουστούν.

Class Car

```
class Car
{

private:
   int _pos;

public:
   void InitializePosition();
   void Move();
   int GetPosition();
   bool Collide(Car)
};
```

Methods

```
void Car::InitializePosition()
{
  _pos = 0;
}

void Car::Move()
{
  _pos += floor((double(rand())/RAND_MAX)*3)-1;
}

int Car::GetPosition()
{
  return _pos;
}

bool Car::Collide(Car other)
{
   return (_pos == other.GetPosition());
}
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;
  carX.InitializePosition();
  carY.InitializePosition();

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Συναρτήσεις

```
#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
         area = width * height / 2.0;
         return area;
}
```

Δηλωση Συνάρτησης

Ορίσματα Συνάρτησης

Επιστρεφόμενη τιμή Συνάρτησης

Παράδειγμα

- Στο αρχείο util.h έχουμε κάποιες χρήσιμες συναρτήσεις που πήραμε από κάποιον άλλον.

```
…….

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

…….
int f(){return 1973}
```

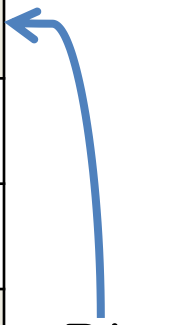
Παράδειγμα (συνέχεια)

```
#include <iostream>
#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 << "English (sqr ml)" << Local::squareArea(2.0);
   cout << PublicUtils::f();
}
```

Δείκτες – Δέσμευση Μνήμης

```
int *aPtr = null;

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

*aPtr = 15;
```

0x1000	15
0x1001	
0x1002	
0x1003	0x1000
0x1004	
0x1005	
0x1006	
0x1007	



Στη C++

```
int *aPtr = new int;

*aPtr = 15;
```

Δείκτες – Αποδέσμευση Μνήμης

```
int *aPtr = null;

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

*aPtr = 15;

free(aPtr);
```

Στη C++

```
int *aPtr = new int;

*aPtr = 15;

delete aPtr;
```

0x1000	15
0x1001	
0x1002	
0x1003	0x0000
0x1004	
0x1005	
0x1006	
0x1007	

aPtr

Δείκτες – Δέσμευση Μνήμης

```
int *aPtr = null;

aPtr = (int *)malloc(3*sizeof(int));

*aPtr = 15; ⇔ aPtr[0] = 15;

*(aPtr + 2) =17; ⇔ aPtr[2] = 17;

free(aPtr)
```

0x1000	
0x1001	
0x1002	
0x1003	0x0000
0x1004	
0x1005	15
0x1006	
0x1007	17

aPtr

Στη C++

```
int *aPtr = new int[3];

*aPtr = 15;
*(aPtr + 2) =17;

delete [] aPtr;
```

Δείκτες - Παράδειγμα

```
int x = 15;

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

*p = x;

cout << p << "\t" << x << endl;

p = &x;

cout << p << "\t" << x << endl;
```

0x1000	
0x1001	
0x1002	
0x1003	
0x1004	
0x1005	
0x1006	
0x1007	

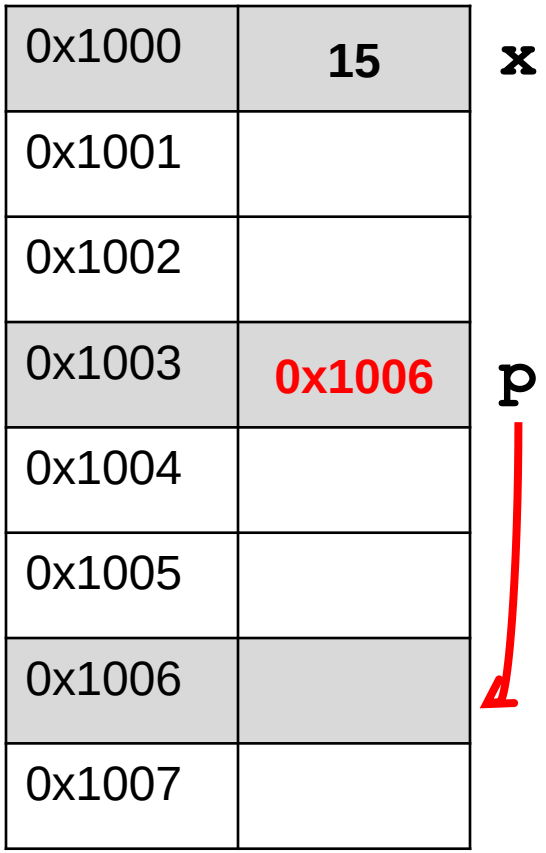
Δείκτες - Παράδειγμα

```
int x = 15;

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

*p = x;

cout << p << " " << x << endl;

p = &x;

cout << p << " " << x << endl;
```

0x1000	15	x
0x1001		
0x1002		
0x1003		
0x1004		
0x1005		
0x1006		
0x1007		

Δείκτες - Παράδειγμα

```
int x = 15;
int *p = (int *)malloc(sizeof(int));
*p = x;
cout << p << "\t" << *p << endl;
p = &x;
cout << p << "\t" << *p << endl;
```



Δείκτες - Παράδειγμα

```
int x = 15;

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

*p = x;

cout << p << "\t" << *p << endl;

p = &x;

cout << p << "\t" << *p << endl;
```

0x1000	**15**	**x**
0x1001		
0x1002		
0x1003	**0x1006**	**p**
0x1004		
0x1005		
0x1006		
0x1007		

Ποιο θα είναι το output?

Δείκτες - Παράδειγμα

```
int x = 15;

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

*p = x;

cout << p << "\t" << *p << endl;

p = &x;

cout << p << "\t" << *p << endl;
```

0x1000	**15**	**x**
0x1001		
0x1002		
0x1003	**0x1006**	**p**
0x1004		
0x1005		
0x1006	**15**	
0x1007		

0x1006 15

Δείκτες - Παράδειγμα

```
int x = 15;

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

*p = x;

cout << p << "\t" << *p << endl;

p = &x;

cout << p << "\t" << *p << endl;
```

0x1000	**15**	**x**
0x1001		
0x1002		
0x1003	**0x1006**	**p**
0x1004		
0x1005		
0x1006	**15**	
0x1007		

Ποιο θα είναι το output?

Δείκτες - Παράδειγμα

```
int x = 15;

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

*p = x;

cout << p << "\t" << *p << endl;

p = &x;

cout << p << "\t" << *p << endl;
```

0x1000	**15**	**x**
0x1001		
0x1002		
0x1003	**0x1000**	**p**
0x1004		
0x1005		
0x1006	**15**	
0x1007		

0x1000 15

Αναφορές (References)

- Μια αναφορά σε μια μεταβλητή είναι σαν ένα συνώνυμο για τη μεταβλητή.

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

- Κατά τη δήλωση πρέπει ΠΑΝΤΑ να καθορίζουμε τη μεταβλητή της οποίας είναι συνώνυμα

Οι αναφορές χρησιμοποιούν τους δείκτες για να αναφερθούν σε μια μεταβλητή. Δεν είναι σαν δείκτες γιατί αναφέρονται μόνιμα σε μια θέση μνήμης και δεν μπορούμε να τις αλλάξουμε.

Τι κρύβουν οι αναφορές

```
int myInt = 5;
int &myRef = myInt;
```
→
```
int * const myRef_p = &myInt;
```

Το = δεν είναι ανάθεση, δημιουργεί την αναφορά.
Δεν μπορούμε να ξανά-αναθέσουμε την αναφορά

```
myInt = 8;
```
→
```
myInt = 8;
```

```
myRef = 7;
```
→
```
*myRef_p = 7;
```

```
cout << myRef;
```
→
```
cout << *myRef_p;
```

Που χρησιμεύουν?

Οι δηλώσεις αναφορών που παίζουν το ρόλο συνώνυμων μιας μεταβλητής σε ένα πρόγραμμα δεν έχουν πολύ μεγάλη χρησιμότητα όπως 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 δια αναφοράς.

Στην περίπτωση που χρησιμοποιήσουμε αναφορές σαν ορίσματα σε συναρτήσεις το πέρασμα δια αναφοράς γίνεται ακόμα πιο απλό.

ΠΕΡΑΣΜΑ ΠΑΡΑΜΕΤΡΩΝ ΣΕ ΣΥΝΑΡΤΗΣΕΙΣ

Πέρασμα δια τιμής

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

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

Τι output θα έχουμε?

Πως δουλεύει

Μνήμη για τη **`main()`**

0x1000	3
0x1001	…
0x1002	…
0x1003	…

`aValue`

Κλήση της **`Increase(aValue)`**

Κλήση της Increase(3)

Μνήμη για τη **`Increase()`**

0x1010	3
0x1011	…
0x1012	…
0x1013	…

`myParameter`

Πως δουλεύει

Μνήμη για τη **`main()`**

0x1000	3
0x1001	…
0x1002	…
0x1003	…

`aValue`

Κλήση της Increase(3)

Μνήμη για τη **`Increase()`**

0x1010	**4**
0x1011	…
0x1012	…
0x1013	…

`myParameter`

```
myParameter ++ ;
```

```
cout << myParameter;
```
output: 4

Πως δουλεύει

Μνήμη για τη **`main()`**

0x1000	3
0x1001	…
0x1002	…
0x1003	…

`aValue`

Επιστροφή από την **`Increase(aValue)`**

`Cout << aValue`
ouptut: 3

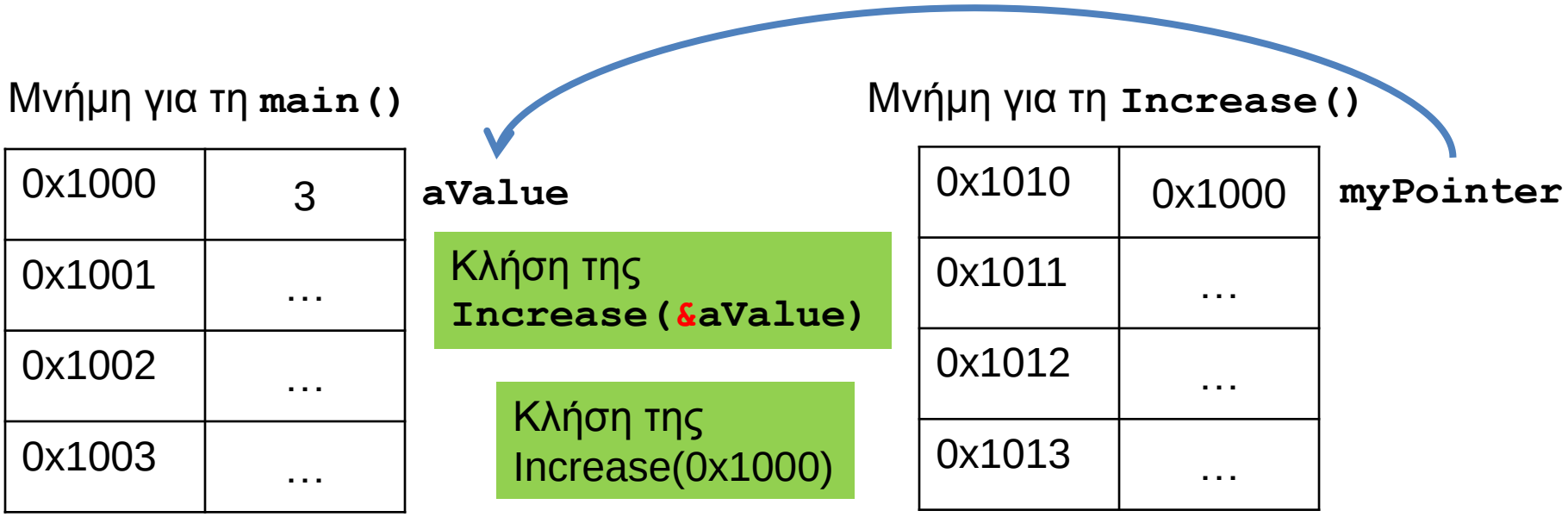
Τελικό output:
4
3

Πέρασμα δια αναφοράς με δείκτη

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

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

Τι output θα έχουμε?

Πως δουλεύει



Πως δουλεύει

Μνήμη για τη **`main()`**

0x1000	**4**
0x1001	…
0x1002	…
0x1003	…

`aValue`

Κλήση της Increase(0x1000)

Μνήμη για τη **`Increase()`**

0x1010	0x1000
0x1011	…
0x1012	…
0x1013	…

`myPointer`

```
(*myPointer)++ ;
```

```
cout << myPointer;
```
output: 4

Πως δουλεύει

Μνήμη για τη **`main()`**

0x1000	4
0x1001	…
0x1002	…
0x1003	…

`aValue`

Επιστροφή από την
`Increase(&aValue)`

`Cout << aValue`
ouptut: 4

Τελικό output:
4
4

Πέρασμα δια αναφοράς με αναφορά

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

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

Χωρίς *

Χωρίς &

Τι output θα έχουμε?

Πως δουλεύει

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

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

Σαν να καλούμε `&myRef = aValue`

Τι κρύβει.

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

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

```
#include <iostream>
void Increase(int *const myRef_p
{
    (*myRef_p)++;
    cout << *myRef_p << "\n";
}
main(){
    int aValue = 3;

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

Το ίδιο αποτέλεσμα όπως με τη χρήση δεικτών!
Ο κώδικας όμως φαίνεται διαφορετικός.

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 και οι αλλαγές ανακλώνται στη συνάρτηση που τις κάλεσε.

- 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);
}
```

Ορίσματα που δεν αλλάζουν

```
#include <iostream>

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

Πίνακες σαν παράμετροι

```
#include <iostream>

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

Πίνακες σαν παράμετροι

```
#include <iostream>

void increment(int arg[], int length){
  for (int i=0;i<length;i++){
        arg[i]++;
  }
}

void printArray(int arg[], int length){
  for (int i =0;i<length;i++){
        cout << arg[i] << endl;
  }
}

main(){
  int anArray[] = {5,10,15};
  incrementArray(anArray,3);
  printArray(anArray,3);
}
```

Τι output θα έχουμε?

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

Συνοπτικά για το πέρασμα παραμέτρων

`function (int array [])`	Η C++ αυτομάτως μετατρέπει τα arrays σε αναφορές
`function (int *var)`	Περνά ένας pointer ως παράμετρος της function

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);
}
```

Τι output θα έχουμε?

```
void f (int x){ …    //could be (int x =1 )
{
  cout << x << endl;
}
```

Default τιμές στο πέρασμα παραμέτρων

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

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

Compile Error!

- στην περίπτωση αυτή σίγουρα a = 10
  - το 5 μπορεί να αντιστοιχεί στο b οπότε c = 20 και d = 0
  - Ή 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 1 <= κ < N πραγματικές παραμέτρους, οι N-κ τελευταίες λαμβάνουν τις default τιμές

ΥΠΕΡΦΟΡΤΩΣΗ (OVERLOADING) ΣΥΝΑΡΤΗΣΕΩΝ ΚΑΙ ΤΕΛΕΣΤΩΝ

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;
}
```

Compile Error!

error: new declaration `float max(int, int)‘ambiguates old declaration `int max(int, int)'

Υπερφόρτωση συναρτήσεων

```
#include <iostream>

using namespace std;

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

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

OK!

Υπερφόρτωση συναρτήσεων

```
int main(){
  int a(3), b(5);
  float x = 1.2, y = 5.4;

  int retInt = max(a,b);
  cout << retInt << endl;

  float retFloat = max(x,y);
  cout << retFloat << endl;
}
```

Άσκηση

- Θέλουμε να φτιάξουμε ένα «παιχνίδι» το οποίο κάνει το εξής:
  - Έχουμε δυο αυτοκίνητα που ξεκινάνε από το σημείο 0 της ευθείας ή από τα σημεία που δίνονται στην είσοδο, και κινούνται τυχαία πάνω στις τιμές των ακεραίων.
    - Σε κάθε κίνηση διαλέγουν τυχαία να πάνε αριστερά, δεξιά, ή να μείνουν στο ίδιο σημείο.
  - Το παιχνίδι σταματάει όταν τα δυο αυτοκίνητα συγκρουστούν.

Class Car

```
class Car
{

private:
   int _pos;

public:
   void InitializePosition();
   void Move();
   int GetPosition();
   bool Collide(Car)
};
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;
  carX.InitializePosition();
  carY.InitializePosition();

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Class Car

```
class Car
{

private:
   int _pos;

public:
   void InitializePosition();
   void InitializePosition(int);
   void Move();
   int GetPosition();
   bool Collide(Car)
};
```

Methods

```
void Car::InitializePosition()
{
   _pos = 0;
}
```

```
void Car::InitializePosition(int p)
{
   _pos = p;
}
```

```
void Car::Move()
{
   _pos += floor((double(rand())/RAND_MAX)*3)-1;
}
```

```
int Car::GetPosition()
{
   return _pos;
}
```

```
bool Car::Collide(Car other)
{
   return (_pos == other.GetPosition());
}
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main(int arc, char ** argv)
{
  Car carX;
  Car carY;
  if (argc == 1){
     carX.InitializePosition();
     carY.InitializePosition();
  }
  else if (argc == 3){
     int x = atoi(argv[1]);
     carX.InitializePosition(x);
     int y = atoi(argv[2]);
     carY.InitializePosition(y);
  }else{
     return;
  }
/* rest of the code */
}
```

Άσκηση

- Θέλουμε να φτιάξουμε ένα «παιχνίδι» το οποίο κάνει το εξής:
  - Έχουμε δυο αυτοκίνητα που ξεκινάνε από το σημείο 0 της ευθείας, και κινούνται πάνω στις τιμές των ακεραίων στο διάστημα [-2,2]
    - Σε κάθε κίνηση μπορούν να πάνε αριστερά, δεξιά, ή να μείνουν στο ίδιο σημείο.
    - Το ένα αυτοκίνητο ελέγχεται από τον χρήστη.
    - Το άλλο κινείται τυχαία.
  - Το παιχνίδι σταματάει όταν τα δυο αυτοκίνητα συγκρουστούν.

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;
  carX.InitializePosition();
  carY.InitializePosition();

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;η
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Πάρε από το input την κίνηση του χρηστή και μετακίνησε το **carX** ανάλογα

Χρειάζεται να υπερφορτώσουμε την μέθοδο **Move** ώστε να παίρνει όρισμα

Class Car

```
class Car
{

private:
   int _pos;

public:
   void InitializePosition();
   void Move();
   void Move(int);
   int GetPosition();
   bool Collide(Car)
};
```

```
void Car::Move()
{
   _pos += floor((double(rand())/RAND_MAX)*3)-1;
   if (_pos > 2){
      _pos = 1;
   }
   if (_pos < -2){
      _pos = -1;
   }
}
```

```
void Car::Move(int inc)
{
   _pos += inc;
   if (_pos > 2){
      _pos = 1;
   }
   if (_pos < -2){
      _pos = -1;
   }
}
```

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;
  carX.InitializePosition();
  carY.InitializePosition();

  bool collision = false;
  int counter = 0;
  while(!collision){
    int inc;
    cin >> inc;
    carX.Move(inc);
    carY.Move();
    cout << "Your car:" << carX.GetPosition()
         << " Other car:" << carY.GetPosition()
          << endl;
    collision = carX.Collide(carY);
    counter ++;
  }
  cout << "Cars collided at position " << carX.GetPosition() << endl;
}
```

Πάρε από το input την κίνηση του χρηστή και μετακίνησε το **carX** ανάλογα

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 μεταβλητές των οποίων ο τύπος ορίστηκε από τον προγραμματιστή.
  - μεταβλητές τύπου struct
  - αντικείμενα κάποιας κλάσης

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);
}
```

Ο τελεστής

Δεσμευμένη λέξη για δήλωση τελεστή

Ο τύπος που επιστρέφει ο τελεστής

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;
}
```

Η ανάθεση δομής Α σε δομή Β αντιγράφει όλα τα πεδία από την Α στη Β.

Υπερφόρτωση τελεστών για κλάσεις

- Για να υπερφορτώσουμε ένα τελεστή ορίσαμε μια συνάρτηση.
- Στην περίπτωση που ο τελεστής επενεργεί πάνω σε αντικείμενα κλάσεων, μπορούμε να ορίσουμε τον τελεστή ως μία μέθοδο της κλάσης.

Ένας διμερής τελεστής καλείται ως μια μέθοδος ενός αντικειμένου που παίρνει ως όρισμα το άλλο αντικείμενο

```
class Complex {
private:
    double re, im;
public:
     double GetRe();
     double GetIm();
     void Set(double, double);
     Complex operator + (Complex)
     int operator == (Complex)
};

Complex operator + (Complex other)
{
    double resultRe = re + other.GetReal();
    double resultIm = re + other.GetIm();
    Complex result;
    result.Set(resultRe, resultIm);
    return result;
}

int operator == (Complex other)
{
    return (re == other.GetReal()) && (im == other.GetIm());
}
```

```
int main ()
{
    Complex a, b;

    a.Set(1.0,2.0);
    b.Set(4.0,1.0);

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

    return 0;
}
```

Η ανάθεση αντικειμένου Α σε αντικείμενο Β αντιγράφει όλα τα πεδία από το Α στο Β.

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;
  carX.InitializePosition();
  carY.InitializePosition();

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = carX.Collide(carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Να αντικαταστήσουμε τη μέθοδο Collide με τον τελεστή ==

```
#include <iostream>
#include <cmath>
#include <cstdlib>

/* Car class definition, method definitions */

int main()
{
  Car carX;
  Car carY;
  carX.InitializePosition();
  carY.InitializePosition();

  bool collision = false;
  int counter = 0;
  while(!collision){
    carX.Move();
    carY.Move();
    collision = (carX == carY);
    counter ++;
  }

  cout << "Cars collided after "  << counter <<" moves "
       << "at position " << carX.GetPosition() << endl;
}
```

Να αντικαταστήσουμε τη μέθοδο Collide με τον τελεστή ==

Class Car

```
class Car
{

private:
   int _pos;

public:
   void InitializePosition();
   void Move();
   int GetPosition();
   bool operator == (Car);
};
```

Methods

```
void Car::InitializePosition()
{
   _pos = 0;
}

void Car::Move()
{
   _pos += floor((double(rand())/RAND_MAX)*3)-1;
}

int Car::GetPosition()
{
   return _pos;
}

bool Car::operator == (Car other)
{
    return (_pos == other.GetPosition());
}
```

Υπερφόρτωση του τελεστή [ ]

- Ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε ένα ασφαλή πίνακα, που ελέγχει πάντα ότι δεν προσπαθούμε να διαβάσουμε ή να γράψουμε σε μια θέση εκτός των ορίων του πίνακα.
- Θα το κάνουμε αυτό υπερφορτώνοντας τον τελεστή [ ]

```
#include <iostream>

using namespace std;

int const SIZE = 100;

class Array{
private:
  int _array[SIZE];
public:
  int operator [] (int i)
  {
    if (i < 0|| i >= SIZE){
      cout << "illegal access\n";
      exit(1);
    }
    return _array[i];
  }
};
```

Προσοχη!! Ο ορισμός αυτός είναι για read-only τελεστή. Πως μπορεί ο τελεστής [ ] να εμφανιστεί στο αριστερό κομμάτι μιας ανάθεσης?

```
int main(){
  Array A;
  A[10] = 1000;
  int x = A[10];
  cout << x;
}
```

```
#include <iostream>

using namespace std;

int const SIZE = 100;

class Array{
private:
  int _array[SIZE];
public:
  int &operator [] (int i)
  {
    if (i < 0|| i >= SIZE){
      cout << "illegal access\n";
      exit(1);
    }
    return _array[i];
  }
};
```

Κάνουμε την συνάρτηση του τελεστή να επιστρέφει μια αναφορά

```
int main(){
  Array A;
  A[10] = 1000;
  int x = A[10];
  cout << x;
}
```

ΔΟΜΕΣ ΚΑΙ ΚΛΑΣΕΙΣ

Δομές (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";
}
```

Κλάσεις και 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";
}
```

Κλάσεις και structs

- Από τους ορισμούς φαίνεται ότι structs και κλάσεις είναι σχεδόν πανομοιότυπες.
  - Υπάρχουν λεπτές διαφορές:
    - Π.χ., by default, οι μεταβλητές μέλη μιας κλάσης είναι private, ενώ τα μέλη ενός struct είναι public.

- Προγραμματιστική πρακτική:
  - Structs χρησιμοποιούνται για την αποθήκευση μόνο δεδομένων (όχι συναρτήσεις), και όλες οι μεταβλητές είναι public.
  - Classes χρησιμοποιούνται για την αποθήκευση δεδομένων και ορισμό μεθόδων. Τα δεδομένα είναι private.

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;
}
```

Όπως και με τις δομές, όταν έχουμε ένα δείκτη σε κλάση, μπορούμε να έχουμε πρόσβαση σε μεταβλητές ή συναρτήσεις μέλη μιας κλάσης χρησιμοποιώντας τον τελεστή **->**

Αναφορές σε αντικείμενα

```
int main ()
{
  complex a;
  complex &ra = a;

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

  return 0;
}
```

Η αναφορά αν και χρησιμοποιεί τον δείκτη για να αναφέρεται στο αντικείμενο χρησιμοποιεί τον τελεστή **.** για να έχει πρόσβαση στα μέλη του αντικείμενου

ΑΛΦΑΡΗΘΜΗΤΙΚΑ

Αλφαριθμητικά (Strings)

- 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,…

- Στη C++ πέρα από την «κλασσική» έννοια του string που προέρχεται από τη C, υπάρχει έτοιμος τύπος δεδομένων που ονομάζεται string που κάνει ακόμα πιο απλή τη διαχείριση τους.
  - 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.
  - Δεν χρειάζεται να ανησυχούμε για το αν η ακολουθία τερματίζεται με '\0'.
  - Τα strings είναι objects και μπορούμε να καλέσουμε συναρτήσεις για επεξεργασία και ιδιότητες του string.

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 και substrings

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

int main(){
 string s1("I saw elvis in a UFO.");
 string s2(s1, 0, 8);
 cout << s2 << endl;

 string s3 = s1 + "Am I crazy?";
 cout << s3 << endl;
 string s4 = s1.substr(2, 11); //(starting index,
 number of chars)
 cout << s4 << endl;
}
```

Strings – Επεξεργασία

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

Strings – Επεξεργασία

```
#include <iostream>
#include <string>
using namespace std;
int main(){
 string s("Hello mother");
 string s1("fa");
 s.replace(6, 2, s1); // starting  pos, how many chars to
  replace, replacement
 cout << s << "\n";
  s.replace(6,2,"bro");
 // if s1.length() > how many chars
 // replace number of chars with the whole of s1
 cout << s << "\n";
}
// start at 7th character, replace 2 chars with the whole of
  "bro"
```

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;

  int found = s.find(s3, start); // find the first occurrence of
 string s3 in string s, starting from start.

  while (found != string::npos){ // if s3 not in s, the function
 returns string::npos (the maximum possible string length)
    s.replace(found, s3.length(), s1);
    start = found + s1.length();
    cout << s << endl;
    found = s.find(s3, start);
  }
  cout << s << endl;
}
```

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_first_not_of("@$.", pos);
// find position of first occurrence of a char NOT in "@$."
  starting from pos

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

s.find_last_not_of("@$.", pos);
// find position of last occurrence of a char NOT in "@$."

const char *p = s.c_str(); // returns a char array from s
```

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
s1.compare(0,2,s2,0,2); // compare s1[0..2]
 with s2[0..2]
s1.swap(s2);
```

alphabetic ordering