Προγραμματισμός Υπολογιστών C++

Βρόχοι Επανάληψης
Πίνακες
Χαρακτήρες

Η Εντολή for

- Η εντολή **for** είναι μία από τις τρεις εντολές επανάληψης και χρησιμοποιείται για τη δημιουργία **επαναληπτικών βρόχων** στη C
- Επαναληπτικός βρόχος καλείται το τμήμα του κώδικα μέσα σε ένα πρόγραμμα, το οποίο εκτελείται από την αρχή και επαναλαμβάνεται όσο μία συνθήκη παραμένει **αληθής (true)**

- Γενική σύνταξη της εντολής **for**:

```
for(αρχική_έκφραση; συνθήκη; τελική_έκφραση)
{
/* ομάδα εντολών (ή αλλιώς "σώμα" του βρόχου) που
   εκτελείται όσο η συνθήκη παραμένει αληθής. */
}
```

Τα Βήματα Εκτέλεσης της `for`

1. Εκτελείται η `αρχική_έκφραση`
   - Η `αρχική_έκφραση` εκτελείται **μόνο μία φορά**, όταν αρχίζει η εκτέλεση της `for` εντολής
   - Συνήθως, είναι μία εντολή εκχώρησης που αρχικοποιεί κάποια μεταβλητή, η οποία θα χρησιμοποιηθεί από τις άλλες δύο εκφράσεις
2. Γίνεται **έλεγχος** της τιμής της `συνθήκης`
   - Η `συνθήκη` είναι συνήθως μία σχεσιακή έκφραση
   - Αν είναι ψευδής, τότε ο `for` βρόχος **τερματίζεται** και η εκτέλεση του προγράμματος συνεχίζει με την **πρώτη εντολή** που υπάρχει **μετά το άγκιστρο κλεισίματος** της `for` εντολής
   - Αν είναι αληθής, τότε εκτελείται η `ομάδα των εντολών` που ονομάζεται και «σώμα του βρόχου»
3. Εκτελείται η `τελική_έκφραση`
   - Συνήθως, η `τελική_έκφραση` αλλάζει την τιμή κάποιας μεταβλητής που χρησιμοποιείται στη `συνθήκη`
4. Επαναλαμβάνονται συνεχώς τα βήματα (2) και (3), μέχρι η τιμή της `συνθήκης` να γίνει ψευδής

Παράδειγμα

```
#include <iostream>
int main()
{
   int i;

   for(i = 0; i < 5; i++)
   {
       std::cout << i << '\n';
   }
   return 0;
}
```

Έξοδος: 0 1 2 3 4

Παρατηρήσεις (1)

- Γενικά, ο βρόχος τερματίζεται είτε όταν η **συνθήκη** γίνει ψευδής ή αν νωρίτερα εκτελεστεί κάποια εντολή που τερματίζει την εκτέλεσή του, όπως η `break` και η `return`
- Όταν **γνωρίζουμε** εκ των προτέρων **τον αριθμό** των επαναλήψεων που επιθυμούμε να εκτελεστούν, τότε χρησιμοποιούμε συνήθως την εντολή `for` και όχι κάποια άλλη επαναληπτική μέθοδο
- Όπως και στην περίπτωση της `if-else` δομής, αν το μπλοκ εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν

Παρατηρήσεις (2)

- Τα τμήματα της `for` εντολής, `αρχική_έκφραση`, `συνθήκη` και `τελική_έκφραση` μπορεί να αποτελούνται από μία μόνο εντολή, αλλά και από περισσότερες
- Στην περίπτωση που αποτελούνται από περισσότερες από μία εντολές, τότε αυτές χωρίζονται μεταξύ τους με τον τελεστή κόμμα (,)

- Π.χ.:

```
for(i = 0, j = 10; i+j<15; i++, j--)
```

`αρχική_έκφραση` `συνθήκη` `τελική_έκφραση`

Παρατηρήσεις (3)

- Ένα πολύ συνηθισμένο λάθος είναι η χρήση του = για να συγκρίνουμε με την τελική τιμή. Π.χ. αν γράψουμε

```
for(i = 0; i = 5; i++)
        std::cout << i << ' ';
```

η συνθήκη είναι πάντα αληθής και, όπως θα δείτε στη συνέχεια, δημιουργείται ένας ατέρμονος βρόχος, ο οποίος εμφανίζει συνεχώς 5, υποβαθμίζοντας σημαντικά την απόδοση του υπολογιστή

- Λάθος είναι επίσης και να γράψετε `i == 5`. Αφού το `i` είναι `0`, η συνθήκη είναι ψευδής και ο βρόχος δεν εκτελείται. Άρα, δεν εμφανίζεται τίποτα στην οθόνη

Παρατηρήσεις (4)

- Όπως και με την **if** εντολή, ένα συνηθισμένο λάθος είναι η αθέλητη πρόσθεση του ερωτηματικού ; στο τέλος της **for** εντολής
- Π.χ. ο παρακάτω κώδικας:

```
for(i = 16; i > 10; i-=3);
      std::cout << i << ' ';
```

κάνει τρεις επαναλήψεις και εμφανίζει μία φορά την τελική τιμή του i που είναι 10

- Συνήθως, **for** βρόχοι με «κενή ομάδα εντολών» χρησιμοποιούνται σαν βρόχοι εισαγωγής χρονικής καθυστέρησης, δηλαδή «για να περάσει η ώρα» μέχρι να γίνει κάποια ενέργεια
- Όταν θέλετε το σώμα του βρόχου να είναι άδειο, συστήνεται το ; να εισάγεται σε ξεχωριστή γραμμή, ώστε να φαίνεται ξεκάθαρα η πρόθεσή σας. Π.χ.

```
for(a = 0; a < 1000; a++)
      ;
```

Παρατηρήσεις (5)

- Σε μία **for** εντολή μπορεί να λείπουν κάποια από τα 3 τμήματά της ή ακόμη και όλα. Π.χ. στην εντολή:

```
for(; a < 5; a++)
```

λείπει η αρχική_έκφραση

- Ωστόσο, το ελληνικό ερωτηματικό ; πρέπει να υπάρχει και να λειτουργεί σαν διαχωριστικό μεταξύ των τμημάτων
- Αν η αρχική_έκφραση και η τελική_έκφραση λείπουν, ο **for** βρόχος είναι ισοδύναμος με τον αντίστοιχο **while** βρόχο, όπως θα δούμε παρακάτω. Π.χ. ο παρακάτω **for** βρόχος:

```
for(; a < 5;)
```

Είναι ισοδύναμος με τον **while** βρόχο:

```
while(a < 5)
```

Παρατηρήσεις (6)

- Όταν σε μία `for` εντολή **λείπει η συνθήκη ή η συνθήκη είναι πάντα αληθής**, τότε αυτός ο `for` βρόχος ονομάζεται **ατέρμονος βρόχος**, δηλαδή, δεν τερματίζεται ποτέ. Π.χ. ο βρόχος:

```
for(a = 0; 0 < 1; a++)
```

είναι ατέρμονος, γιατί η συνθήκη `0 < 1` είναι πάντα αληθής

- Η συνήθης πρακτική για τη δημιουργία ατέρμονου βρόχου είναι να παραλείπονται και οι τρεις εκφράσεις, δηλαδή:

```
for(;;)
```

Παράδειγμα (1)

- Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <iostream>
int main()
{
 int i = 0, j = 5;
 for(i > j; i+j == 5; j < 2)
 {
        std::cout << "One\n";
        i = 4;
        j = 2;
 }
 std::cout << i << ' ' << j;
 return 0;
}
```

Παράδειγμα (1)

- Απάντηση. Η τιμή της έκφρασης `i > j` είναι `0`, αλλά αυτό δεν επηρεάζει την εκτέλεση του βρόχου. Ας δούμε αναλυτικά τις επαναλήψεις του.

  1η επανάληψη. Αφού το `i` είναι `0`, η συνθήκη `i+j == 5` (`0+5 = 5 == 5`) είναι αληθής και επομένως εμφανίζεται το `One`. Στη συνέχεια, οι τιμές των `i` και `j` γίνονται `4` και `2`, αντίστοιχα

  2η επανάληψη. Η τιμή της έκφρασης `j < 2` είναι `0`, αλλά δεν επηρεάζει την εκτέλεση του βρόχου. Η συνθήκη `i+j == 5` (`4+2 = 6 == 5`) γίνεται ψευδής και τερματίζεται η εκτέλεση του βρόχου

  Το πρόγραμμα εμφανίζει τις τιμές των `i` και `j` που είναι `4` και `2`, αντίστοιχα.

Παράδειγμα (2)

- Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <iostream>
int main()
{
 int i;
 unsigned int j;

 for(i = 12; i > 2; i -= 5)
       std::cout << i << ' ';
       std::cout << "End = " << i << '\n';

 for(j = 2; j >= 0; j--)
       std::cout << "Test\n";

 return 0;
}
```

Παράδειγμα (2)

- Απάντηση. Αφού η `for` εντολή δεν έχει άγκιστρα, ο μεταγλωττιστής θεωρεί ότι το σώμα του βρόχου αποτελείται από μία μόνο εντολή, δηλαδή από την πρώτη `cout`. Η δεύτερη `cout`, παρά την παραπλανητική στοίχισή της, εκτελείται μόνο μία φορά, μετά το τέλος του βρόχου. Άρα, αρχικά εμφανίζεται `12 7 End = 2`

  Πάμε στην επόμενη παγίδα. Αφού το `j` έχει δηλωθεί σαν απρόσημος ακέραιος, η τιμή του δεν θα γίνει ποτέ αρνητική. Το `-1` θα μετατραπεί σε μία unsigned τιμή (π.χ. σε 32-bit σύστημα είναι 4294967295). Επομένως, ο δεύτερος βρόχος είναι ατέρμονος και το πρόγραμμα εμφανίζει συνέχεια `Test`

Παράδειγμα (3)

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει έναν ακέραιο και, αν αυτός ανήκει στο [10, 20], να εμφανίζει τόσες φορές όσες και η τιμή του αριθμού τη λέξη `One`, αλλιώς να διαβάζει 10 ακεραίους και να εμφανίζει πόσους αρνητικούς αριθμούς εισήγαγε ο χρήστης

Παράδειγμα (3)

```
#include <iostream>
using namespace std;
int main()
{
	int i, num, neg;

	cout << "Enter number: ";
	cin >> num;

	if(num >= 10 && num <= 20)
	{
		for(i = 0; i < num; i++)
			cout << "One\n";
	}
	else
	{
		neg = 0;
		for(i = 0; i < 10; i++)
		{
			cout << "Enter number: ";
			cin >> num;
			if(num < 0)
				neg++;
		}
		cout << "Negatives = " << neg << '\n';
	}
	return 0;
}
```

Η Εντολή break

- Η εντολή **break** εκτός από τον τερματισμό της **switch** εντολής, μπορεί να χρησιμοποιηθεί για τον άμεσο τερματισμό ενός **for**, **while** ή **do-while** βρόχου

- Στους επαναληπτικούς βρόχους, μετά την εκτέλεση της εντολής **break** το πρόγραμμα **συνεχίζει** με την **εκτέλεση της πρώτης** εντολής **μετά** τον βρόχο

- Όπως θα δούμε στη συνέχεια, η εκτέλεση της εντολής **break** μέσα σε έναν **ένθετο** επαναληπτικό βρόχο προκαλεί τον τερματισμό μόνο του βρόχου στον οποίο η ίδια περιέχεται

- Επίσης, όπως είδαμε στην εντολή **switch**, η εκτέλεση της εντολής **break** μέσα σε μία **switch** προκαλεί επίσης τον άμεσο τερματισμό της λειτουργίας της

Παράδειγμα

```
#include <iostream>
int main()
{
   int i;

   for(i = 1; i < 10; i++)
   {
       if(i == 5)
             break;

       std::cout << i << ' ';
   }
   std::cout << "\nOut: " << i << '\n';
   return 0;
}
```

Παράδειγμα

Έξοδος: `1 2 3 4`

`Out = 5`

Η Εντολή continue

- Η εντολή **continue** χρησιμοποιείται μόνο μέσα σε έναν **for**, **while** ή **do-while** επαναληπτικό βρόχο

- Η εκτέλεση της εντολής **continue** τερματίζει την τρέχουσα επανάληψη του βρόχου που την περιέχει και προκαλεί την έναρξη της επόμενης επανάληψης

- Άρα, οι εντολές μετά την εντολή **continue** **δεν εκτελούνται** για την τρέχουσα επανάληψη

Παράδειγμα

```
#include <iostream>
int main()
{
   int i;

   for(i = 0; i < 5; i++)
   {
       if(i == 2 || i == 3)
             continue;
       std::cout << i << ' ';
   }
   return 0;
}
```

Παράδειγμα

Έξοδος: `0 1 4`

Ένθετοι Βρόχοι

- Ένας επαναληπτικός βρόχος (π.χ. **for**, **while** ή **do-while**) μπορεί να είναι **ένθετος** στο εσωτερικό κάποιου άλλου
- Οποιοσδήποτε βρόχος μπορεί να είναι ένθετος μέσα σε οποιοδήποτε άλλο είδος βρόχου (π.χ. **while** βρόχος μέσα σε **for** βρόχο)
- Π.χ. στην παρακάτω γενική περίπτωση, βλέπουμε δύο ένθετα **for**, στα οποία για να συμβεί μία επανάληψη του εξωτερικού βρόχου πρέπει πρώτα να τερματίσει η εκτέλεση του εσωτερικού βρόχου

Εξωτερικός for βρόχος

Εσωτερικός for βρόχος

```
for(αρχική_έκφραση_1; συνθήκη_1; τελική_έκφραση_1)
{
   for(αρχική_έκφραση_2; συνθήκη_2; τελική_έκφραση_2)
   {
         /* ομάδα εντολών που θα εκτελείται συνεχώς
         όσο η συνθήκη_2 παραμένει αληθής. */
   }
   /* ομάδα εντολών που θα εκτελείται συνεχώς όσο η
  συνθήκη_1 παραμένει αληθής. */
}
```

Παράδειγμα (1)

```
#include <iostream>
int main()
{
   int i, j;

   for(i = 0; i < 2; i++)
   {
       std::cout << "One ";
       for(j = i+1; j < 2; j++)
             std::cout << "Two ";
   }
   return 0;
}
```

Παράδειγμα (1)

Έξοδος: `One Two One`

Παράδειγμα (2)

- Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <iostream>
int main()
{
   int i, j;

   for(i = 1; i < 3; i++)
   {
        for(j = 0; j < i; j++)
        {
               if(i+j == 1)
                      break;
               std::cout << "Two ";
        }
        std::cout << "One ";
   }
   std::cout << i << ' ' << j << '\n';
   return 0;
}
```

Παράδειγμα (2)

Έξοδος: `One Two Two One 3 2`

Η Εντολή while

- Γενική σύνταξη της εντολής while:

```
while(συνθήκη)
{
/* ομάδα εντολών που θα εκτελείται όσο η
  συνθήκη παραμένει αληθής. */
}
```

Εκτέλεση της Εντολής while

1. Γίνεται έλεγχος της τιμής της συνθήκης
   - Αν η συνθήκη είναι ψευδής (false) τότε ο while βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της while εντολής
   - Αν η συνθήκη είναι αληθής (true) τότε εκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα {} και η τιμή της συνθήκης ελέγχεται πάλι
     - Αν η τιμή της συνθήκης γίνει ψευδής (false), τότε ο while βρόχος τερματίζεται
     - Αν όχι, επανεκτελείται η ομάδα των εντολών του βρόχου while

   Η παραπάνω διαδικασία επαναλαμβάνεται μέχρι η τιμή της συνθήκης να γίνει ψευδής

Παρατηρήσεις (1)

- Η εντολή **while** χρησιμοποιείται συνήθως όταν **δεν γνωρίζουμε τον ακριβή αριθμό** των επαναλήψεων που θέλουμε να εκτελεστεί η ομάδα των εντολών μας
  - Όταν αντιθέτως **γνωρίζουμε** εκ των προτέρων **τον αριθμό** των επαναλήψεων που επιθυμούμε να εκτελεστούν, τότε συνήθως χρησιμοποιούμε την εντολή **for**

- Όπως και σε προηγούμενες περιπτώσεις (π.χ. εντολές **if-else**, **for**, κτλ), αν η ομάδα εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν

- Όπως και με την εντολή **for**, το ; στο τέλος της **while** εντολής δηλώνει ότι το σώμα του βρόχου δεν περιέχει καμία εντολή. Π.χ. ο παρακάτω βρόχος γίνεται ατέρμονος

```
int a = 1;
while(a != 10);
 a++; // Δεν θα εκτελεστεί ποτέ
```

Παρατηρήσεις (2)

- Η εντολή `while(x)` είναι ισοδύναμη με την εντολή `while(x != 0)` και η `while(x == 0)` ισοδύναμη με την `while(!x)`

- Αν η συνθήκη είναι **πάντα αληθής**, ο βρόχος θα εκτελείται συνεχώς, εκτός αν περιέχει μία εντολή που να τον τερματίσει (π.χ. `break`)

- Π.χ. ο βρόχος `while(1)` είναι ατέρμονος, γιατί η συνθήκη είναι πάντα αληθής, αφού το 1 είναι διαφορετικό από το 0

Παρατηρήσεις (3)

- Οι εντολές `for` και `while` είναι πολύ στενά συνδεδεμένες
- Ουσιαστικά, οι εντολές είναι ισοδύναμες (εκτός αν ο `for` βρόχος περιέχει την εντολή `continue`):

```
for(exp1; exp2; exp3)
{
  /* εντολές */
}
```

```
exp1;
while(exp2)
{
    /* εντολές */
    exp3;
}
```

Παράδειγμα

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς έναν ακέραιο και να εμφανίζει τη λέξη `Hi` τόσες φορές όσες και η τιμή του ακεραίου. Αν ο χρήστης εισάγει αρνητικό αριθμό, η εισαγωγή των ακεραίων να σταματάει και το πρόγραμμα να εμφανίζει τον συνολικό αριθμό των `Hi` που εμφανίστηκαν στην οθόνη. Χρησιμοποιήστε μόνο `while` βρόχους

Παράδειγμα

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

int main()
{
	int i, num, times;

	times = 0;
	while(1)
	{
		cout << "Enter number: ";
		cin >> num;
		if(num < 0)
			break;
		i = 0;
		while(i < num)
		{
			cout << "Hi\n";
			i++;
		}
		times += num;
	}
	cout << "Total number is: " << times << '\n';
	return 0;
}
```

Η Εντολή do-while

- Αντίθετα με τις εντολές **for** και **while** όπου η συνθήκη ελέγχου ελέγχεται πριν από την εκτέλεση των εντολών, η εντολή **do-while** ελέγχει τη συνθήκη ελέγχου μετά την εκτέλεση των εντολών
- Επομένως, ο βρόχος **do-while** εκτελείται τουλάχιστον μία φορά
- Γενική σύνταξη της εντολής **do-while**:

```
do
{
/* ομάδα εντολών που εκτελείται αρχικά μία φορά
   και στη συνέχεια κατ' επανάληψη όσο η συνθήκη
   παραμένει αληθής. */
} while(συνθήκη);
```

Τα Βήματα Εκτέλεσης της do-while

1. Εκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα {}

2. Γίνεται έλεγχος της τιμής της συνθήκης
   - Αν η συνθήκη είναι ψευδής (false) τότε ο do-while βρόχος τερματίζεται και η εκτέλεση του προγράμματος συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της do-while εντολής
   - Αν η συνθήκη είναι αληθής (true) τότε επανεκτελείται η ομάδα εντολών που υπάρχει ανάμεσα στα άγκιστρα {}
   - Το βήμα αυτό επαναλαμβάνεται μέχρι η τιμή της συνθήκης να γίνει ψευδής

Παρατηρήσεις (1)

- Γενικά, ο βρόχος **do-while** χρησιμοποιείται λιγότερο από τους **for** και **while** βρόχους, αφού μπορεί να αντικατασταθεί από αυτούς
- Μία πολύ συνηθισμένη χρήση του είναι για έλεγχο εγκυρότητας των τιμών που εισάγει ο χρήστης
- Ο βρόχος **do-while** **πρέπει να τελειώνει** με το ελληνικό ερωτηματικό (;)

Παρατηρήσεις (2)

- Σε περίπτωση που ο βρόχος **do-while** περιέχει μόνο μία εντολή, τα άγκιστρα μπορούν, και στην **do-while** εντολή, να παραλειφθούν

- Παρόλα αυτά, προτείνεται **να χρησιμοποιείτε πάντοτε τα άγκιστρα σε εντολές do-while**, διότι η παράλειψή τους πιθανότατα να παραπλανήσει τους αναγνώστες του προγράμματός σας (στο «κάτω» κομμάτι της εντολής), κάνοντάς τους να νομίζουν ότι χρησιμοποιείτε **while** εντολή

Π.χ.:

```
do
  std::cout << i << '\n';
while(++i <= 10);
```

Παράδειγμα

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει συνεχώς έναν ακέραιο και να εμφανίζει τη λέξη `Hi` τόσες φορές όσες και η τιμή του ακεραίου. Το πρόγραμμα να υποχρεώνει τον χρήστη να εισάγει έναν θετικό ακέραιο. Αν ο χρήστης εισάγει διαδοχικά την ίδια τιμή το πρόγραμμα να τερματίζει. Χρησιμοποιήστε μόνο `do-while` βρόχους

Παράδειγμα (2)

```
#include <iostream>
using std::cout;
using std::cin;
int main()
{
    int i, num, last;

    do
    {
          cout << "Enter number: ";
          cin >> num;
    } while(num <= 0);

    do
    {
          last = num;
          i = 1;
          do
          {
                    cout << "Hi\n";
                    i++;
          } while(i <= num);

          do
          {
                    cout << "Enter number: ";
                    cin >> num;
          } while(num <= 0);
    } while(last != num);
    return 0;
}
```

Η Εντολή goto

- Με την εντολή goto μπορούμε να μεταφέρουμε τον έλεγχο του προγράμματος σε κάποια άλλη εντολή μέσα στην ίδια συνάρτηση, με την προϋπόθεση ότι η εντολή έχει μία ετικέτα

- Γενική σύνταξη της εντολής goto:

```
goto label;
```

- Όταν εκτελείται η εντολή goto η εκτέλεση του προγράμματος μεταβαίνει άμεσα στην εντολή που ακολουθεί τη θέση που έχει δηλωθεί με το όνομα label

- Η θέση με το όνομα label ονοματίζεται όπως και μία μεταβλητή και πρέπει να είναι μοναδική μέσα στη συνάρτηση όπου χρησιμοποιείται η εντολή goto

- Το όνομά της πρέπει να ακολουθείται από την άνω κάτω τελεία :

Παράδειγμα

- Αν ο χρήστης εισάγει την τιμή -1 η εκτέλεση του προγράμματος μεταβαίνει στη θέση START και ο for βρόχος εκτελείται πάλι από την αρχή

```
#include <iostream> // Παράδειγμα 6.12
int main()
{
  int i, num;
START:
  for(i = 0; i < 5; i++)
  {
   std::cout << "Enter number: ";
   std::cin >> num;
   if(num == -1)
          goto START; /* Θα μπορούσαμε να γράψουμε i=-1 και
  να έχουμε το ίδιο αποτέλεσμα, απλά είναι παράδειγμα για
  τη χρήση της goto. */
  }
  return 0;
}
```

Παρατηρήσεις (1)

- Γενικά, δε συστήνεται η χρήση της goto, γιατί η μετάβαση της εκτέλεσης του προγράμματος από ένα σημείο σε κάποιο άλλο και μετά σε κάποιο άλλο δημιουργεί δυσνόητο κώδικα που δεν είναι καλά οργανωμένος και άρα δύσκολα διαβάζεται και ελέγχεται
- Πολλοί μάλιστα είναι εντελώς αντίθετοι στη χρήση της goto, υποστηρίζοντας ότι δεν έχει καμία θέση μέσα σε ένα καλά δομημένο πρόγραμμα
- Ωστόσο, υπάρχουν περιπτώσεις που η goto μπορεί να φανεί χρήσιμη, όπως για την αποφυγή επανάληψης του ίδιου κώδικα με την ομαδοποίηση κοινών εντολών. Π.χ.

```
if(error_1)
        goto ERROR;
...
if(error_2)
        goto ERROR;
...
ERROR:
... // Εντολές διαχείρισης ανεπιθύμητων καταστάσεων.
```

Παρατηρήσεις (2)

- Σαν ένα ακόμα παράδειγμα, επειδή η **break** τερματίζει μόνο το **switch** ή τον βρόχο στον οποίο περιέχεται, η **goto** μπορεί να φανεί χρήσιμη για την άμεση έξοδο από ένα ένθετο **switch**, ή από ένα **switch** μέσα σε ένα βρόχο, ή από έναν ένθετο βρόχο. Π.χ.

```
for(i = 0; i < 10; i++)
        for(j = 0; j < 20; j++)
                for(k = 0; k < 30; k++)
                {
                        if(συνθήκη)
                                goto NEXT;
                }
NEXT:
...
```

- Αν και γενικά πρέπει να αποφεύγετε την χρήση της, να θυμάστε ότι κάποιες φορές μπορεί να φανεί χρήσιμη και να οδηγήσει σε απλούστερο και πιο ευανάγνωστο κώδικα. Άλλωστε, και οι εντολές **break**, **continue** και **return** είναι παραλλαγές της **goto**

Πίνακες

- Ένας πίνακας είναι **μία δομή δεδομένων** η οποία περιέχει έναν συγκεκριμένο αριθμό στοιχείων **του ίδιου τύπου** (π.χ. πίνακας ακεραίων αριθμών, πίνακας πραγματικών αριθμών, πίνακας χαρακτήρων, ...)

- Κάθε στοιχείο μπορεί να προσπελαστεί μέσω της θέσης του στον πίνακα

- Όλοι οι πίνακες **δεσμεύουν συνεχόμενες θέσεις στη μνήμη** (στην περιοχή μνήμης που ονομάζεται **στοίβα** ή **stack**) του υπολογιστή

- Συνηθέστερα είδη είναι οι μονοδιάστατοι και οι διδιάστατοι πίνακες

Δήλωση Μονοδιάστατου Πίνακα

- Για να δηλώσουμε έναν μονοδιάστατο πίνακα πρέπει να καθορίσουμε το όνομα του πίνακα, τον τύπο δεδομένων των στοιχείων του πίνακα και το πλήθος των στοιχείων του πίνακα

- Η γενική περίπτωση δήλωσης ενός μονοδιάστατου πίνακα είναι:

```
τύπος_δεδομένων  όνομα_πίνακα[πλήθος_στοιχείων_πίνακα];
```

Παρατηρήσεις

- Το όνομα_πίνακα πρέπει να είναι μοναδικό (να μην υπάρχει άλλη μεταβλητή στο πρόγραμμα με το ίδιο όνομα)
- Το πλήθος_στοιχείων_πίνακα αλλιώς και μήκος, μέγεθος ή διάσταση του πίνακα, δηλώνεται από μία θετική σταθερή ακέραια έκφραση μέσα σε αγκύλες [ ]
- Όλα τα στοιχεία του πίνακα έχουν τον ίδιο τύπο δεδομένων, ο οποίος μπορεί να είναι οποιοσδήποτε τύπος (π.χ. int, float, char, …)

Δέσμευση Μνήμης (1)

- Όταν δηλώνεται ένας πίνακας, ο μεταγλωττιστής δεσμεύει ένα τμήμα μνήμης για να αποθηκεύσει τις τιμές των στοιχείων του
  - Αυτές οι τιμές αποθηκεύονται η μία μετά την άλλη, σε διαδοχικές θέσεις μνήμης
  - Τυπικά, αυτό το κομμάτι μνήμης δεσμεύεται από συγκεκριμένο μέρος της μνήμης που ονομάζεται **στοίβα** (**stack**), και αποδεσμεύεται όταν τερματιστεί η λειτουργία της συνάρτησης μέσα στην οποία έχει δηλωθεί ο πίνακας

- Π.χ., με την παρακάτω δήλωση ο μεταγλωττιστής δεσμεύει 40 bytes για να αποθηκεύσει τις τιμές των 10 ακεραίων στοιχείων του

```
int arr[10];
```

- Για την εύρεση του μεγέθους της δεσμευμένης μνήμης από έναν πίνακα, μπορούμε να χρησιμοποιήσουμε τον τελεστή `sizeof` (π.χ., `sizeof(arr)`)

Δέσμευση Μνήμης (2)

- Το μέγιστο μέγεθος μνήμης που μπορεί να δεσμευτεί με τη δήλωση ενός πίνακα εξαρτάται από τη διαθέσιμη μνήμη στη στοίβα
- Π.χ. το επόμενο πρόγραμμα μπορεί να μην εκτελεστεί σε κάποιον υπολογιστή, αν δεν υπάρχει η διαθέσιμη μνήμη στη στοίβα για την αποθήκευση των τιμών

```
#include <iostream>
int main()
{
        double arr[500000];
        return 0;
}
```

Θα δούμε στη συνέχεια ότι μπορούμε να δεσμεύσουμε μνήμη και από μία μεγαλύτερη περιοχή που ονομάζεται σωρός (heap)

- Όταν η μνήμη είναι πολύτιμη, μη δηλώνετε πίνακες με μέγεθος μεγαλύτερο από ότι χρειάζεται, ώστε να αποφεύγεται η άσκοπη σπατάλη της

Στοιχεία Μονοδιάστατου Πίνακα

- Για να αναφερθούμε σε κάποιο στοιχείο του πίνακα γράφουμε το **όνομα του πίνακα** συνοδευόμενο από τον **δείκτη θέσης** του στοιχείου μέσα σε αγκύλες `[]`
- **Ο δείκτης θέσης** πρέπει να είναι μία ακέραια σταθερά, μεταβλητή ή έκφραση, η οποία **προσδιορίζει τη θέση** του συγκεκριμένου στοιχείου στον πίνακα
- Το πρώτο στοιχείο ενός πίνακα με μέγεθος `n` στοιχεία αποθηκεύεται στη θέση `[0]` του πίνακα, το δεύτερο στοιχείο στη θέση `[1]`, το τρίτο στη θέση `[2]`, ... κ.ο.κ., με αποτέλεσμα το τελευταίο στοιχείο να αποθηκεύεται στη θέση `[n-1]`
- Κάθε στοιχείο μπορεί να χρησιμοποιηθεί όπως μία απλή μεταβλητή. Π.χ:

```
int i, j, a[10], b[10];
a[0] = 2; // Η τιμή του πρώτου στοιχείου γίνεται 2.
b[9] = a[0]; // Η τιμή του τελευταίου στοιχείου γίνεται 2.
i = j = b[9]+1; // Οι τιμές των i και j γίνονται 3.
a[i+j] = 100; /* Αφού i+j=3+3=6, η τιμή του έβδομου στοιχείου γίνεται
100. */
b[2*i-1] = a[i%j]; /* Η τιμή του b[5] γίνεται ίση με του a[0], δηλαδή
2. */
```

Παρατηρήσεις (1)

- Το πλήθος των στοιχείων του πίνακα δεν μπορεί να αλλάξει κατά την εκτέλεση του προγράμματος. Δηλαδή, δεν μπορείτε να προσθέσετε νέα στοιχεία στον πίνακα ή να διαγράψετε στοιχεία από αυτόν

- Αν το μήκος του πίνακα χρησιμοποιείται πολλές φορές μέσα στο πρόγραμμα, μία καλή πρακτική είναι να το αντικαταστήσετε με μία σταθερά. Αν στο μέλλον χρειαστεί να αλλάξετε το μήκος του, απλά αλλάζετε την τιμή της σταθεράς στο σημείο της δήλωσής της

Παρατηρήσεις (2)

- Να αποφεύγετε παρενέργειες στη δεικτοδότηση των στοιχείων, π.χ. μη γράψετε κάτι σαν αυτό:

```
b[i] = a[i++];
```

  - Εξαρτάται από τον μεταγλωττιστή πότε θα γίνει η αύξηση του `i`, δηλαδή πριν ή μετά την ανάθεση

- Μην ξεχνάτε ότι η αρίθμηση των στοιχείων ενός πίνακα `n` στοιχείων ξεκινά απ' το `0` (όχι απ' το `1`) και φτάνει ως και το `n-1`. Κάποτε είχα διαβάσει αυτό:

"My girlfriend told me that I care about programming more than about her. I told her that in an array of my interests she is [1] - she was satisfied"

- Για την αντιγραφή ενός πίνακα (π.χ. `int a[10]`) σε έναν άλλον (π.χ. `int b[10]`), δεν επιτρέπεται να γράψετε κάτι σαν το παρακάτω:

```
b = a;
```

παρόλο που φαίνεται αρκετά «κομψό». Θα καταλάβετε γιατί, όταν εξηγήσουμε τη στενή σχέση μεταξύ πινάκων και δεικτών στην επόμενη διάλεξη

Παρατηρήσεις (3)

- Ένα ακόμα λάθος συμβαίνει όταν θέλουμε να ελέγξουμε αν οι δύο πίνακες έχουν τις ίδιες τιμές. Φαίνεται λογικό να γράψουμε `if(b == a)`. Όχι, αυτή η σύγκριση δεν λειτουργεί όπως θα περιμένατε, δεν συγκρίνονται τα στοιχεία των πινάκων. Θα δούμε στην επόμενη διάλεξη τι πραγματικά συγκρίνεται. Και το χειρότερο είναι ότι αυτή η παραπλανητική συνθήκη μεταγλωττίζεται και μας δίνει την εσφαλμένη εντύπωση ότι πράγματι οι πίνακες συγκρίνονται. Προσοχή, είναι ένα πολύ συνηθισμένο λάθος

Παρατηρήσεις (4)

- **Προσοχή!!!** Η C++ δεν ελέγχει αν κάνετε υπέρβαση των ορίων των θέσεων του πίνακα, αλλά αφήνει τον προγραμματιστή υπεύθυνο γι' αυτό... Σε περίπτωση, όμως, που γίνει υπέρβαση των ορίων του πίνακα, η συμπεριφορά του προγράμματος είναι απρόβλεπτη
- Π.χ. δείτε το διπλανό πρόγραμμα

```
#include <iostream>
int main()
{
        int i, j = 20, arr[3];

        for(i = 0; i < 4; i++)
                arr[i] = 100;

        std::cout << j;
        return 0;
}
```

- Ο `arr` περιέχει τρία στοιχεία και οι επιτρεπτές τιμές των δεικτών θέσης είναι από 0 έως 2
- Στην τελευταία επανάληψη (`i=3`) η έκφραση `arr[3] = 100;` αναθέτει μία τιμή σε ένα στοιχείο που δεν ανήκει στον πίνακα και συγκεκριμένα την τιμή 100 που υπερεγγράφει (overwrites) το περιεχόμενο της μνήμης ακριβώς μετά το στοιχείο `arr[2]`
- Αν η συγκεκριμένη μνήμη έχει δεσμευτεί για τη μεταβλητή `j`, το `j` αλλάζει τιμή και το πρόγραμμα θα εμφανίσει 100 αντί για 20...!!!

Δήλωση Πίνακα και Απόδοση Αρχικών Τιμών (1)

- Τη δήλωση του πίνακα την ακολουθεί ο τελεστής = και οι τιμές των στοιχείων διαχωρίζονται με κόμμα (,) μέσα σε άγκιστρα {}. Π.χ. με τη δήλωση `int arr[4] = {10, 20, 30, 40};` οι τιμές των `arr[0]`, `arr[1]`, `arr[2]`, `arr[3]` γίνονται `10, 20, 30, 40` αντίστοιχα
- Αν η λίστα των τιμών είναι μικρότερη από το πλήθος των στοιχείων του πίνακα, τα υπόλοιπα στοιχεία αρχικοποιούνται με την προεπιλεγμένη τιμή που έχει ο τύπος του στοιχείου. Η προεπιλεγμένη τιμή για ακέραιους είναι το 0 και για πραγματικούς τύπους το 0.0.
- Π.χ. με τη δήλωση `int arr[4] = {10, 20};` οι τιμές των `arr[0]` και `arr[1]`, γίνονται `10` και `20`, ενώ οι τιμές των `arr[2]` και `arr[3]` γίνονται `0`
- Η λίστα των τιμών δεν επιτρέπεται να είναι μεγαλύτερη από το πλήθος των στοιχείων του πίνακα

Δήλωση Πίνακα και Απόδοση Αρχικών Τιμών (2)

- Αν το μήκος του πίνακα παραληφθεί, ο μεταγλωττιστής δημιουργεί έναν πίνακα με μέγεθος ίσο με το πλήθος των τιμών στη λίστα. Π.χ. με τη δήλωση `int arr[4] = {10, 20, 30, 40};` ο μεταγλωττιστής δημιουργεί έναν πίνακα τεσσάρων ακεραίων και θέτει τις τιμές 10, 20, 30 και 40 στα στοιχεία του
- Αν θέλουμε οι τιμές ενός πίνακα, μονοδιάστατου ή πολυδιάστατου, να παραμένουν οι ίδιες κατά την εκτέλεση του προγράμματος, δηλώνουμε τον πίνακα σαν `const`. Ένας `const` πίνακας πρέπει να αρχικοποιηθεί όταν δηλώνεται
- Π.χ. `const int arr[4] = {10, 20, 30, 40};` Ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους σε οποιαδήποτε προσπάθεια αλλαγής της τιμής κάποιου στοιχείου, όπως π.χ. με την εντολή: `arr[0] = 80;`

Παράδειγμα (1)

- Να γραφεί ένα πρόγραμμα το οποίο να δηλώνει έναν πίνακα 5 πραγματικών και με χρήση επαναληπτικού βρόχου να θέτει τις τιμές 1.1, 1.2, 1.3, 1.4 και 1.5 στα στοιχεία του. Στη συνέχεια, να εμφανίζει τα στοιχεία του πίνακα με αντίστροφη σειρά, δηλαδή από το τελευταίο προς το πρώτο

Παράδειγμα (1)

```
#include <iostream>
int main()
{
   int i;
   double arr[5];

   for(i = 0; i < 5; i++)
       arr[i] = 1.1 + (i*0.1);

   for(i = 4; i >= 0; i--)
       std::cout << arr[i] << '\n';
   return 0;
}
```

Παράδειγμα (2)

- Ποια είναι η έξοδος του παρακάτω προγράμματος;

```
#include <iostream>
int main()
{
        int i, a[] = {20, 10, 0, -10, -20};
        for(i = 0; a[i]; i++)
                if(a[i] < 0)
                        std::cout << a[i] << ' ';
        return 0;
}
```

- Ποια είναι τα περιεχόμενα του πίνακα **a** στον παρακάτω κώδικα;

```
int i, a[] = {4, 2, 0}, b[] = {2, 3, 4};
for(i = 0; i < 3; i++)
        a[b[i]-a[2-i]]++;
```

Παράδειγμα (2)

1. Η συνθήκη `a[i]` στην `for` εντολή ισοδυναμεί με `a[i] != 0`. Αφού η τιμή του τρίτου στοιχείου είναι `0`, ο βρόχος θα τερματιστεί, άρα το πρόγραμμα δεν εμφανίζει τίποτα

2. Τα `a[0]`, `a[1]`, και `a[2]` γίνονται `5`, `3`, και `1`, αντίστοιχα

Παράδειγμα (3)

- Να γραφεί ένα πρόγραμμα το οποίο να δηλώνει δύο πίνακες 10 ακεραίων και να διαβάζει τις τιμές των στοιχείων τους. Στη συνέχεια, να ελέγχει αν υπάρχουν κοινά στοιχεία στους δύο πίνακες και, αν ναι, να εμφανίζει την τιμή του κάθε κοινού στοιχείου. Αλλιώς, να εμφανίζει ένα μήνυμα ότι δεν υπάρχουν κοινά στοιχεία

Παράδειγμα (3)

```
#include <iostream>
using std::cout;
using std::cin;
int main()
{
    const int SIZE = 10;
    int i, j, cnt, arr1[SIZE], arr2[SIZE];

    for(i = 0; i < SIZE; i++)
    {
          cout << "Enter number_" << i+1 << " for the 1st array: ";
          cin >> arr1[i];
          cout << "Enter number_" << i+1 << " for the 2nd array: ";
          cin >> arr2[i];
    }
    cnt = 0; /* Μεταβλητή που μετράει τα κοινά στοιχεία. */
    for(i = 0; i < SIZE; i++)
    {
          for(j = 0; j < SIZE; j++) /* Αυτός ο βρόχος ελέγχει αν το στοιχείο του πρώτου πίνακα
    υπάρχει στον δεύτερο. */
          {
                    if(arr1[i] == arr2[j])
                    {
                              cnt++;
                              cout << arr1[i] << '\n';
                    }
          }
    }
    if(cnt == 0)
          cout << "No common elements were found\n";
    return 0;
}
```

Η for εύρους Εντολή

- Η C++11 εισάγει μία πιο απλή μορφή της for εντολής για την προσπέλαση μίας ακολουθίας τιμών, όπως σε έναν πίνακα ή σε έναν αποδέκτη (π.χ. vector). Αυτή η μορφή της for εντολής ονομάζεται for εύρους (range-for)
- Η σύνταξή της είναι: for(δήλωση : έκφραση)

όπου η έκφραση μετά την : αντιπροσωπεύει μία ακολουθία τιμών και η δήλωση καθορίζει τη μεταβλητή που θα χρησιμοποιηθεί για να αποθηκευτούν οι τιμές των στοιχείων της ακολουθίας. Π.χ.

```
int arr[] = {10, 20, 30, 40};
for(auto i : arr)
        std::cout << i << '\n';
```

Σε κάθε επανάληψη, από το πρώτο μέχρι και το τελευταίο στοιχείο του πίνακα, το i γίνεται ίσο με το αντίστοιχο στοιχείο του πίνακα, δηλαδή, η τιμή του αντιγράφεται στο i, και αυτή εμφανίζεται στην οθόνη. Ο βρόχος διασχίζει όλα τα στοιχεία του πίνακα. Δηλαδή, το i γίνεται διαδοχικά ίσο με 10, 20, 30, 40

- Θα μπορούσαμε να δηλώσουμε το i σαν int, απλά με την λέξη auto, η επανάληψη γράφεται με γενικό τρόπο που να μην εξαρτάται από τον τύπο των στοιχείων του πίνακα. Είναι πιο ευέλικτο και βολικό να αφήσουμε τον μεταγλωττιστή να συμπεράνει τον τύπο

Η Πρότυπη Κλάση `vector` (1)

- Η πρότυπη κλάση `vector` είναι ένας αποδέκτης που χρησιμοποιείται για την αποθήκευση στοιχείων του ίδιου τύπου σε συνεχόμενες θέσεις μνήμης
- Η κλάση `vector` είναι μία από τις πιο χρήσιμες και ευρέως χρησιμοποιούμενες κλάσεις της Standard Template Library (STL)
- Σε αντίθεση με τους πίνακες, μπορούμε να καθορίσουμε το αρχικό μέγεθος ενός `vector` αντικειμένου δυναμικά, και το σημαντικότερο, το μέγεθός του μπορεί να αυξηθεί δυναμικά ανάλογα με τις απαιτήσεις του προγράμματος
- Δηλαδή, η κλάση `vector` υποστηρίζει δυναμική διαχείριση μνήμης που επιτρέπει την προσθήκη ή τη διαγραφή στοιχείων κατά την εκτέλεση του προγράμματος
- Γενικά, η διαχείριση της κλάσης `vector` είναι σχετικά απλή και παρέχει πολλές δυνατότητες, μεγαλύτερη ασφάλεια και ευελιξία από ότι ο πίνακας

Η Πρότυπη Κλάση vector (2)

- Για να χρησιμοποιήσουμε την κλάση vector πρέπει να συμπεριλάβουμε το αρχείο vector. Για να δημιουργήσουμε ένα vector αντικείμενο πρέπει να καθορίσουμε τον τύπο των στοιχείων που θα περιέχει. Π.χ, ας δούμε μερικά παραδείγματα δήλωσης vector αντικειμένων

```
vector<int> arr; // Δημιουργία άδειου διανύσματος ακεραίων στοιχείων.
vector<int>  arr(3,  20);  /* Δημιουργία διανύσματος τριών ακεραίων
στοιχείων με αρχική τιμή 20. */
vector<int>  arr{1,  2,  3,  4};  /* Δημιουργία διανύσματος τεσσάρων
ακεραίων στοιχείων με τις αντίστοιχες αρχικές τιμές. */
int num;
cin >> num;
vector<double>  arr(num);  /* Δημιουργία διανύσματος num πραγματικών
στοιχείων. */
```

Παράδειγμα

- Το παρακάτω πρόγραμμα διαβάζει 10 αριθμούς και αποθηκεύει σε ένα vector αντικείμενο, που αρχικά είναι άδειο, τους θετικούς αριθμούς. Στη συνέχεια εμφανίζει τις τιμές των στοιχείων που έχουν αποθηκευτεί

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        int i, num;
        vector<int> vec;
        for(i = 0; i < 10; i++)
        {
                cout << "Enter number: ";
                cin >> num;
                if(num > 0)
                        vec.push_back(num);
        }
        for(i = 0; i < vec.size(); i++)
                cout << vec[i] << '\n';
        return 0;
}
```

Διδιάστατοι Πίνακες

- Οι **διδιάστατοι πίνακες** μοιάζουν με τους γνωστούς μαθηματικούς πίνακες δύο διαστάσεων της άλγεβρας και αποτελούνται και αυτοί από **γραμμές** και **στήλες**

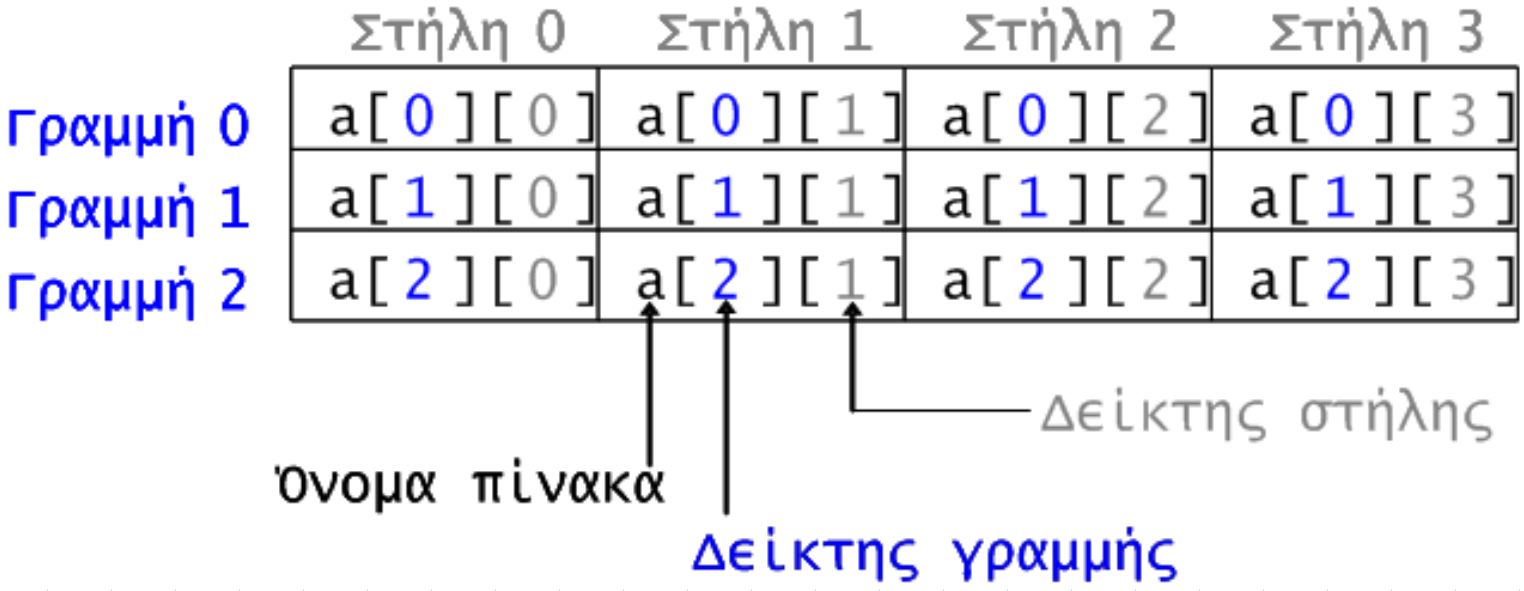
- Για να δηλώσουμε έναν διδιάστατο πίνακα πρέπει να καθορίσουμε το όνομα του πίνακα, τον τύπο δεδομένων των στοιχείων του πίνακα, καθώς και το πλήθος των γραμμών και των στηλών του

```
τύπος_δεδομένων όνομα[πλήθος_γραμμών][πλήθος_στηλών]
```

- Το **πλήθος των στοιχείων** ενός διδιάστατου πίνακα είναι ίσο με το **γινόμενο** του πλήθους **των γραμμών** του επί το πλήθος **των στηλών** του

Στοιχεία Διδιάστατου Πίνακα

- Π.χ. η εντολή `int a[3][4];` δηλώνει έναν διδιάστατο πίνακα με όνομα `a`, ο οποίος περιέχει 12 στοιχεία και καθένα από αυτά τα στοιχεία είναι ένας ακέραιος αριθμός (`int`)
- Για να αναφερθούμε σε κάποιο στοιχείο γράφουμε το όνομα του πίνακα και τους δείκτες γραμμής και στήλης μέσα σε διπλές αγκύλες `[][]`. Όπως και με τους μονοδιάστατους πίνακες, η αρίθμηση των δεικτών θέσης γραμμής και στήλης αρχίζει από το μηδέν. Π.χ.



Διδιάστατοι Πίνακες και Μνήμη (1)

- Όπως και με τους μονοδιάστατους πινάκες, όταν δηλώνεται ένας διδιάστατος πίνακας, ο μεταγλωττιστής δεσμεύει ένα τμήμα μνήμης από τη στοίβα για να αποθηκεύσει τα στοιχεία του
- Τα στοιχεία αποθηκεύονται στη μνήμη διαδοχικά ανά γραμμή, αρχίζοντας πρώτα με τα στοιχεία της πρώτης γραμμής, μετά της δεύτερης, της τρίτης, κ.ο.κ.
- Αν και μιλάμε για πολυδιάστατους πίνακες, στην πραγματικότητα, η C++ υποστηρίζει μόνο μονοδιάστατους πίνακες. Απλά, επειδή το στοιχείο ενός πίνακα μπορεί και αυτό να είναι πίνακας μπορούμε να προσομοιώσουμε πολυδιάστατους πίνακες
- Π.χ., τα στοιχεία του πίνακα `a` είναι τα `a[0]`, `a[1]` και `a[2]`, όπου το καθένα από αυτά είναι πίνακας τεσσάρων ακεραίων

Διδιάστατοι Πίνακες και Μνήμη (2)

- Θεωρήστε έναν διδιάστατο πίνακα, έστω `int a[ROWS][COLS]`
- Αφού τα στοιχεία του πίνακα αποθηκεύονται σε διαδοχικές θέσεις μνήμης, ο μεταγλωττιστής για να βρει τη διεύθυνση μνήμης ενός στοιχείου (και δεδομένου επίσης ότι το όνομα του πίνακα ισούται με τη διεύθυνση του πρώτου στοιχείου του πίνακα, όπως θα δούμε σε επόμενη διάλεξη), ο μεταγλωττιστής για να βρει τη διεύθυνση μνήμης του στοιχείου `a[i][j]`

  α) υπολογίζει το μέγεθος σε bytes της γραμμής, δηλ: `row_size = COLS × sizeof(int)` και το πολλαπλασιάζει με το `i`

  β) πολλαπλασιάζει το `j` με το μέγεθος σε bytes του ενός στοιχείου και

  γ) προσθέτει τα δύο γινόμενα στη διεύθυνση του πρώτου στοιχείου του πίνακα, άρα:

```
memory_address = a + (i × row_size) + (j × sizeof(int))
```

- Παρατηρήστε ότι για τον υπολογισμό της διεύθυνσης μνήμης ενός στοιχείου, μόνο ο αριθμός των στηλών είναι απαραίτητος

Αρχικοποίηση Διδιάστατου Πίνακα (1)

- Όπως και ένας μονοδιάστατος πίνακας, ένας διδιάστατος πίνακας μπορεί να αρχικοποιηθεί μαζί με τη δήλωσή του. Οι τιμές εκχωρούνται στα στοιχεία του πίνακα ανά γραμμή, ξεκινώντας από τα στοιχεία της πρώτης γραμμής, μετά της δεύτερης, κ.ο.κ. Π.χ. με τη δήλωση:

```
int arr[3][3] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};
```

η τιμή του `arr[0][0]` γίνεται 10, η τιμή του `arr[0][1]` γίνεται 20, η τιμή του `arr[0][2]` γίνεται 30 κ.ο.κ. Εναλλακτικά, μπορούμε να παραλείψουμε τα εσωτερικά άγκιστρα και να γράψουμε:

```
int arr[3][3] = {10, 20, 30, 40, 50, 60, 70, 80, 90};
```

- Με τα εσωτερικά άγκιστρα, αν η λίστα των τιμών είναι μικρότερη από το πλήθος των στοιχείων μίας γραμμής, ο μεταγλωττιστής εκχωρεί την τιμή 0 στα υπόλοιπα στοιχεία της. Αν είναι μεγαλύτερη, είναι λάθος. Π.χ. με τη δήλωση:

```
int arr[3][3] = {{10, 20}, {40, 50}, {70}};
```

οι τιμές των `arr[0][2]` , `arr[1][2]` , `arr[2][1]` και `arr[2][2]` αρχικοποιούνται με 0

Αρχικοποίηση Διδιάστατου Πίνακα (2)

- Αν παραλείψουμε την αρχικοποίηση κάποιας γραμμής, ο μεταγλωττιστής εκχωρεί την τιμή 0 στα στοιχεία της. Π.χ. με τη δήλωση:

  ```
  int arr[3][3] = {{10, 20, 30}};
  ```

  τα στοιχεία της δεύτερης και της τρίτης γραμμής γίνονται 0

- Αν παραλείπονται τα εσωτερικά άγκιστρα και η λίστα των τιμών είναι μικρότερη από το πλήθος των στοιχείων του πίνακα, ο μεταγλωττιστής αποδίδει την τιμή 0 στα υπόλοιπα στοιχεία. Π.χ. με τη δήλωση:

  ```
  int arr[3][3] = {10, 20};
  ```

  οι τιμές των `arr[0][0]` και `arr[0][1]` γίνονται 10 και 20 αντίστοιχα, ενώ όλων των υπολοίπων στοιχείων γίνονται 0

- Όταν δηλώνεται ένας διδιάστατος πίνακα, ο αριθμός των στηλών πρέπει υποχρεωτικά να καθοριστεί. Ο αριθμός των γραμμών είναι προαιρετικός. Αν δεν δηλωθεί, ο μεταγλωττιστής θα δημιουργήσει έναν διδιάστατο πίνακα με βάση τη λίστα αρχικοποίησης. Π.χ. με τη δήλωση:

  ```
  int arr[][3] = {10, 20, 30, 40, 50, 60};
  ```

  αφού ο πίνακας έχει τρεις στήλες και οι αρχικές τιμές είναι έξι, ο μεταγλωττιστής θα δημιουργήσει έναν διδιάστατο πίνακα ακεραίων με δύο γραμμές και τρεις στήλες

Αρχικοποίηση Διδιάστατου Πίνακα (3)

- Αν η αρχική τιμή μπορεί εύκολα να παραχθεί ή είναι η ίδια για όλα τα στοιχεία, ένας συνήθης τρόπος αρχικοποίησης του πίνακα είναι με τη χρήση διπλών επαναληπτικών βρόχων. Π.χ. ο επόμενος κώδικας αρχικοποιεί τα στοιχεία του πίνακα με την τιμή 60

```
int row, col, arr[5][10];
for(row = 0; row < 5; row++)
      for(col = 0; col < 10; col++)
            arr[row][col] = 60;
```

Παράδειγμα

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει οκτώ ακεραίους, να τους αποθηκεύει σε έναν πίνακα **2*4** και να εμφανίζει τα στοιχεία του πίνακα αντίστροφα, δηλαδή ξεκινώντας από το «κάτω-δεξιά» στοιχείο και καταλήγοντας στο «πάνω-αριστερά»

```
int row, col, arr[5][10];
for(row = 0; row < 5; row++)
      for(col = 0; col < 10; col++)
            arr[row][col] = 60;
```

Παράδειγμα

```
#include <iostream>
using namespace std;
int main()
{
    const int ROWS = 2;
    const int COLS = 4;
    int i, j, arr[ROWS][COLS];

    for(i = 0; i < ROWS; i++)
    {
            for(j = 0; j < COLS; j++)
            {
                    cout << "Enter arr[" << i << "][" << j << "]: ";
                    cin >> arr[i][j];
            }
    }
    cout << "\nArray elements\n";
    cout << "--------------\n";
    for(i = ROWS-1; i >= 0; i--)
    {
            for(j = COLS-1; j >= 0; j--)
                    cout << arr[i][j] << ' ';
            cout << '\n';
    }
    return 0;
}
```

Χαρακτήρες

- Έως τώρα έχουμε κατά κύριο λόγο χρησιμοποιήσει τους αριθμητικούς τύπους δεδομένων `int`, `float` και `double`
- Ο τύπος δεδομένων `char` είναι κι αυτός **αριθμητικός**
- Για τη διαχείριση των χαρακτήρων (και των αλφαριθμητικών στο επόμενο κεφάλαιο), θα θεωρήσουμε ότι το σύνολο των χαρακτήρων που υποστηρίζει ο υπολογιστής είναι κωδικοποιημένο σύμφωνα με το πιο διαδεδομένο πρότυπο, τον **κώδικα ASCII**, που **αντιστοιχίζει** κάθε χαρακτήρα σε μία **αριθμητική τιμή**
- Ο **ASCII κώδικας** αντιστοιχίζει (κωδικοποιεί) ένα σύνολο χαρακτήρων που αποτελείται από γράμματα, αριθμούς, σημεία στίξης, κτλ... με ακέραιες τιμές ανάμεσα στο `0` και το `255`
- Π.χ. η ASCII τιμή του χαρακτήρα `'C'` είναι το `67`, ενώ η ASCII τιμή του χαρακτήρα `'c'` είναι το `99`

Πίνακας ASCII - Βασικοί χαρακτήρες (0-127)

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
(nul)	0	0x00	(sp)	32	0x20	@	64	0x40	`	96	0x60
(soh)	1	0x01	!	33	0x21	A	65	0x41	a	97	0x61
(stx)	2	0x02	"	34	0x22	B	66	0x42	b	98	0x62
(etx)	3	0x03	#	35	0x23	C	67	0x43	c	99	0x63
(eot)	4	0x04	$	36	0x24	D	68	0x44	d	100	0x64
(enq)	5	0x05	%	37	0x25	E	69	0x45	e	101	0x65
(ack)	6	0x06	&	38	0x26	F	70	0x46	f	102	0x66
(bel)	7	0x07	'	39	0x27	G	71	0x47	g	103	0x67
(bs)	8	0x08	(	40	0x28	H	72	0x48	h	104	0x68
(ht)	9	0x09	)	41	0x29	I	73	0x49	i	105	0x69
(nl)	10	0x0a	*	42	0x2a	J	74	0x4a	j	106	0x6a
(vt)	11	0x0b	+	43	0x2b	K	75	0x4b	k	107	0x6b
(np)	12	0x0c	,	44	0x2c	L	76	0x4c	l	108	0x6c
(cr)	13	0x0d	-	45	0x2d	M	77	0x4d	m	109	0x6d
(so)	14	0x0e	.	46	0x2e	N	78	0x4e	n	110	0x6e
(si)	15	0x0f	/	47	0x2f	O	79	0x4f	o	111	0x6f
(dle)	16	0x10	0	48	0x30	P	80	0x50	p	112	0x70
(dc1)	17	0x11	1	49	0x31	Q	81	0x51	q	113	0x71
(dc2)	18	0x12	2	50	0x32	R	82	0x52	r	114	0x72
(dc3)	19	0x13	3	51	0x33	S	83	0x53	s	115	0x73
(dc4)	20	0x14	4	52	0x34	T	84	0x54	t	116	0x74
(nak)	21	0x15	5	53	0x35	U	85	0x55	u	117	0x75
(syn)	22	0x16	6	54	0x36	V	86	0x56	v	118	0x76
(etb)	23	0x17	7	55	0x37	W	87	0x57	w	119	0x77
(can)	24	0x18	8	56	0x38	X	88	0x58	x	120	0x78
(em)	25	0x19	9	57	0x39	Y	89	0x59	y	121	0x79
(sub)	26	0x1a	:	58	0x3a	Z	90	0x5a	z	122	0x7a
(esc)	27	0x1b	;	59	0x3b	[	91	0x5b	{	123	0x7b
(fs)	28	0x1c	<	60	0x3c	\	92	0x5c	|	124	0x7c
(gs)	29	0x1d	=	61	0x3d	]	93	0x5d	}	125	0x7d
(rs)	30	0x1e	>	62	0x3e	^	94	0x5e	~	126	0x7e
(us)	31	0x1f	?	63	0x3f	_	95	0x5f	(del)	127	0x7f

Πίνακας ASCII - Επιπλέον χαρακτήρες (128-255)

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
Ç	128	0x80	á	160	0xa0	└	192	0xc0	α	224	0xe0
ü	129	0x81	í	161	0xa1	┴	193	0xc1	ß	225	0xe1
é	130	0x82	ó	162	0xa2	┬	194	0xc2	Γ	226	0xe2
â	131	0x83	ú	163	0xa3	├	195	0xc3	π	227	0xe3
ä	132	0x84	ñ	164	0xa4	─	196	0xc4	Σ	228	0xe4
à	133	0x85	Ñ	165	0xa5	┼	197	0xc5	σ	229	0xe5
å	134	0x86	ª	166	0xa6	╞	198	0xc6	µ	230	0xe6
ç	135	0x87	º	167	0xa7	╟	199	0xc7	τ	231	0xe7
ê	136	0x88	¿	168	0xa8	╚	200	0xc8	Φ	232	0xe8
ë	137	0x89	⌐	169	0xa9	╔	201	0xc9	Θ	233	0xe9
è	138	0x8a	¬	170	0xaa	╩	202	0xca	Ω	234	0xea
ï	139	0x8b	½	171	0xab	╦	203	0xcb	δ	235	0xeb
î	140	0x8c	¼	172	0xac	╠	204	0xcc	∞	236	0xec
ì	141	0x8d	¡	173	0xad	═	205	0xcd	φ	237	0xed
Ä	142	0x8e	«	174	0xae	╬	206	0xce	ε	238	0xee
Å	143	0x8f	»	175	0xaf	╧	207	0xcf	∩	239	0xef
É	144	0x90	░	176	0xb0	╨	208	0xd0	≡	240	0xf0
æ	145	0x91	▒	177	0xb1	╤	209	0xd1	±	241	0xf1
Æ	146	0x92	▓	178	0xb2	╥	210	0xd2	≥	242	0xf2
ô	147	0x93	│	179	0xb3	╙	211	0xd3	≤	243	0xf3
ö	148	0x94	┤	180	0xb4	╘	212	0xd4	⌠	244	0xf4
ò	149	0x95	╡	181	0xb5	╒	213	0xd5	⌡	245	0xf5
û	150	0x96	╢	182	0xb6	╓	214	0xd6	÷	246	0xf6
ù	151	0x97	╖	183	0xb7	╫	215	0xd7	≈	247	0xf7
ÿ	152	0x98	╕	184	0xb8	╪	216	0xd8	°	248	0xf8
Ö	153	0x99	╣	185	0xb9	┘	217	0xd9	∙	249	0xf9
Ü	154	0x9a	║	186	0xba	┌	218	0xda	·	250	0xfa
¢	155	0x9b	╗	187	0xbb	█	219	0xdb	√	251	0xfb
£	156	0x9c	╝	188	0xbc	▄	220	0xdc	ⁿ	252	0xfc
¥	157	0x9d	╜	189	0xbd	▌	221	0xdd	²	253	0xfd
Pts	158	0x9e	╛	190	0xbe	▐	222	0xde	■	254	0xfe
ƒ	159	0x9f	┐	191	0xbf	▀	223	0xdf		255	0xff

Ο Τύπος char (1)

- Για να αποθηκεύσουμε ένα χαρακτήρα σε μία μεταβλητή χρησιμοποιούμε τον τύπο **char**
- Βέβαια, επειδή ο χαρακτήρας κωδικοποιείται σαν ακέραιος, μπορούμε να χρησιμοποιήσουμε και κάποιον άλλο ακέραιο τύπο (π.χ. **int**)
- Στο επόμενο παράδειγμα δηλώνεται μία μεταβλητή τύπου **char** με το όνομα **ch** και αποθηκεύεται ο χαρακτήρας **'a'** σε αυτήν

```
char ch;
ch = 'a';
```

- Όταν χρησιμοποιείται κάποιος σταθερός χαρακτήρας πρέπει να περικλείεται σε **μονές αποστρόφους** (**' '**) και **όχι** σε διπλά εισαγωγικά

Ο Τύπος char (2)

- Όταν αποθηκεύεται ένας χαρακτήρας σε μία μεταβλητή τύπου **char**, στην πραγματικότητα αποθηκεύεται η ASCII τιμή του χαρακτήρα. Δηλαδή, στο προηγούμενο παράδειγμα στη μεταβλητή **ch** αποθηκεύτηκε η τιμή **97**

- Επομένως, οι εντολές: **ch = 'a';** και **ch = 97;** είναι ισοδύναμες

- Βέβαια, είναι προτιμότερο να αποφεύγετε τη χρήση της αριθμητικής τιμής, όχι μόνο γιατί το πρόγραμμά σας θα διαβάζεται πιο εύκολα, αλλά και γιατί δεν θα εξαρτάται από το σύνολο χαρακτήρων που χρησιμοποιείται

- Παρατηρήστε ότι οι πιο συνηθισμένοι χαρακτήρες, όπως **γράμματα**, **ψηφία** και **σημεία στίξης** αντιστοιχίζονται σε αριθμητικές τιμές ανάμεσα στο **0** και το **127**

- Οι χαρακτήρες με τιμές από **128** έως **255** αποτελούν το εκτεταμένο ASCII σύνολο και αντιστοιχίζονται σε εξεζητημένα γράμματα και ειδικά σύμβολα

Χρήση Χαρακτήρα

- Για να εμφανίσουμε την ASCII τιμή με το cout προσαρμόζουμε τον τύπο της μεταβλητής σε int.Π.χ.

```
char ch = 'a';
cout << ch << ' ' << (int)ch;
```

Ο κώδικας εμφανίζει a 97

- Ουσιαστικά, όταν ένας χαρακτήρας περιέχεται σε μία έκφραση, είτε είναι σταθερά είτε μεταβλητή, η C++ τον χειρίζεται σαν ακέραιο και χρησιμοποιεί την ASCII τιμή του

- Αφού η C++ χειρίζεται τους χαρακτήρες σαν ακεραίους, μπορούμε να τους χρησιμοποιήσουμε σε αριθμητικές εκφράσεις. Π.χ.

```
char ch = 'c';
int i;
ch++; /* Η μεταβλητή ch γίνεται 'd'. */
ch = 68; /* Η μεταβλητή ch γίνεται 'D'. */
i = ch-3; /* Η μεταβλητή i γίνεται 'A' δηλ. 65. */
```

Παρατηρήσεις

- Επειδή η μέγιστη τιμή που μπορεί να πάρει μία μεταβλητή `char` είναι το 127 (θυμηθείτε ότι το εύρος τιμών του `char` είναι -128…127) σε περίπτωση που θέλουμε να αποθηκεύσουμε σε μία μεταβλητή `char` ένα χαρακτήρα με ASCII τιμή μεγαλύτερη από 127, πρέπει να χρησιμοποιήσουμε κάποιον άλλον τύπο, όπως π.χ. `unsigned char` ή `int`

- Ένα συνηθισμένο σημείο που «μπερδεύει» στην αρχή είναι η διαχείριση χαρακτήρων που αντιστοιχούν σε ψηφία. Για παράδειγμα, η ASCII τιμή του χαρακτήρα 2 δεν είναι 2, αλλά 50. Αν θέλουμε να χρησιμοποιήσουμε τον χαρακτήρα 2 γράφουμε `'2'`, όχι 2. Παρόμοια, για να ελέγξουμε αν ένας χαρακτήρας αντιστοιχεί σε ψηφίο 0-9 γράφουμε:

```
if(ch >= '0' && ch <= '9') όχι if(ch >= 0 && ch <= 9)
```

Διάβασμα Χαρακτήρων

- Υπάρχουν πολλοί τρόποι για να διαβαστεί ένας χαρακτήρας από το ρεύμα εισόδου
- Ο πιο συνηθισμένος τρόπος είναι με την `get()`, η οποία είναι συνάρτηση μέλος της `istream` κλάσης
- Η `get()` αρχίζει να διαβάζει χαρακτήρες όταν ο χρήστης πατήσει *Enter*. Αν εκτελεστεί επιτυχημένα, επιστρέφει τον χαρακτήρα που διαβάστηκε σαν `int`
- Αν δεν υπάρχουν άλλοι διαθέσιμοι χαρακτήρες για διάβασμα, επιστρέφει μία ειδική σταθερά που δηλώνεται στο `iostream` με το όνομα `EOF` (*End Of File*)
- Για παράδειγμα, αν ο χρήστης πληκτρολογήσει τους χαρακτήρες `abc` και πατήσει *Enter*, η πρώτη κλήση της get() επιστρέφει το `'a'`, η δεύτερη το `'b'`, η τρίτη το `'c'` και η τέταρτη το `'\n'`. Αν κληθεί πάλι, το πρόγραμμα «κολλάει» μέχρι ο χρήστης να εισάγει νέο(υς) χαρακτήρα(ες) και πατήσει πάλι *Enter*

Παράδειγμα (1)

- Το παρακάτω πρόγραμμα εμφανίζει και μετράει τους χαρακτήρες που εισάγει ο χρήστης μέχρι να πατήσει *Enter*

```
#include <iostream>
#include <cstdio>
using std::cin;
using std::cout;

int main()
{
  int ch, sum;

  cout << "Enter characters: ";
  sum = 0;
  while((ch = cin.get()) != '\n' && ch != EOF) /* Επειδή ο τελεστής != έχει
μεγαλύτερη προτεραιότητα από τον = χρησιμοποιούμε παρενθέσεις. */
  {
            sum++;
            cout << (char)ch; /* Προσαρμόζουμε τον τύπο για την εμφάνιση του
χαρακτήρα. */
  }
  cout << '\n' << sum << " characters are read\n";
  return 0;
}
```

Η `get()` επιστρέφει έναν-έναν τους χαρακτήρες που εισήγαγε ο χρήστης μέχρι να συναντήσει τον `'\n'` χαρακτήρα. Κάθε φορά, τον αποθηκεύει στην `ch` και τον συγκρίνει με την `EOF` τιμή για να ελέγξει ότι έχει έγκυρη τιμή

Παράδειγμα (2)

- Μία πολύ συνηθισμένη περίπτωση που κάνει πολλούς να αναρωτιούνται γιατί το πρόγραμμά τους δεν λειτουργεί σωστά είναι όταν μετά το διάβασμα αριθμητικών τιμών ακολουθεί διάβασμα χαρακτήρων. Το παρακάτω πρόγραμμα αποτελεί ένα τέτοιο παράδειγμα. Εντοπίζετε κάποια **δυσλειτουργία** όταν ο χρήστης εισάγει έναν ακέραιο και πατήσει *Enter*;

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

int main()
{
  int i, ch;

  cout << "Enter number: ";
  cin >> i;

  cout << "Enter character: ";
  ch = cin.get();
  cout << i << ' ' << (char)ch << '\n';
  return 0;
}
```

Παράδειγμα (2)

- Απάντηση. Όταν ο χρήστης εισάγει έναν ακέραιο και πατήσει *Enter*, ο χαρακτήρας `'\n'` που δημιουργείται αποθηκεύεται στην ουρά εισόδου. Η `get()` τον παίρνει από εκεί, τον εκχωρεί στη μεταβλητή `ch` και έτσι δεν αφήνει τον χρήστη να εισάγει κάποιο χαρακτήρα. Άρα, το πρόγραμμα εμφανίζει μόνο τον ακέραιο. Υπάρχουν διάφοροι τρόποι για να ξεφορτωθούμε τον χαρακτήρα `'\n'`. Ένας τρόπος είναι να προσθέσουμε άλλη μία εντολή `get()` πριν από τη δεύτερη `cout`. Δηλαδή, `cin.get();` Επίσης, αν αντιστρέψουμε τη σειρά διαβάσματος, δηλαδή, πρώτα διαβάσουμε τον χαρακτήρα και μετά τον ακέραιο, το πρόγραμμα θα εκτελεστεί σωστά, αφού οι λευκοί χαρακτήρες, όπως ο `'\n'`, παραλείπονται πριν από μία αριθμητική τιμή

Παράδειγμα (3)

- Να γραφεί ένα πρόγραμμα το οποίο να εμφανίζει όλα τα πεζά γράμματα του λατινικού αλφαβήτου σε μία γραμμή, τα κεφαλαία γράμματα σε μία δεύτερη γραμμή και τους χαρακτήρες που αντιστοιχούν στα ψηφία 0-9 σε μία τρίτη γραμμή. Να χρησιμοποιήσετε μόνο έναν `for` βρόχο

Παράδειγμα (3)

```
#include <iostream>
int main()
{
  char ch, end_ch;

  end_ch = 'z';
  for(ch = 'a'; ch <= end_ch; ch++)
  {
          std::cout << ch << ' ';
          if(ch == 'z')
          {
                  ch = 'A'-1; /* Αφαιρούμε 1, ώστε στην επόμενη
επανάληψη του βρόχου με την εντολή ch++ να γίνει ίση με 'A'. */
                  end_ch = 'Z'; /* Αλλάζουμε τον τερματικό χαρακτήρα,
ώστε ο βρόχος να εμφανίσει τα κεφαλαία γράμματα. */
                  std::cout << '\n';
          }
          else if(ch == 'Z')
          {
                  ch = '0'-1;
                  end_ch = '9';
                  std::cout << '\n';
          }
  }
  return 0;
}
```