C++

Μεταβλητές, Σταθερές και Αριθμητικές Μετατροπές

Είσοδος/Έξοδος Δεδομένων

Μεταβλητές

- **Σχέση Μνήμης Υπολογιστή και Μεταβλητών**
  - Η μνήμη (RAM) ενός υπολογιστή αποτελείται από πολλά εκατομμύρια θέσεις αποθήκευσης δεδομένων που έχουν διαδοχική αρίθμηση
  - Το μέγεθος κάθε θέσης μνήμης είναι μία οκτάδα (byte)
  - Π.χ. σκεφτείτε ότι ένας παλιός υπολογιστής με μόνο 16MB μνήμης έχει μνήμη:

    16 * 1.024 = 16.384 kbytes

    16.384 * 1.024 = 16.777.216 θέσεις μνήμης (bytes)
  - Κάθε θέση μνήμης μπορεί να έχει ένα όνομα και ένα περιεχόμενο
  - Μία **μεταβλητή** ονομάζεται μία **θέση μνήμης** που έχει ένα συγκεκριμένο **όνομα** και **περιεχόμενο** (π.χ. `i`)
  - Η **τιμή** μίας **μεταβλητής** είναι **το περιεχόμενο** αυτής της θέσης μνήμης (π.χ. 10)

Ονόματα Μεταβλητών

- **Απαράβατοι κανόνες κατά τη δήλωση του ονόματος μίας μεταβλητής**
  - Το όνομα μπορεί να περιέχει γράμματα, ψηφία και τον χαρακτήρα υπογράμμισης _
  - Ο πρώτος χαρακτήρας πρέπει να είναι γράμμα ή ο χαρακτήρας υπογράμμισης _
  - Η γλώσσα C++ είναι **case sensitive** (δηλ. κάνει διάκριση μεταξύ των πεζών και κεφαλαίων γραμμάτων)
    - Συνεπώς, η μεταβλητή με το όνομα `sum` είναι διαφορετική από τη μεταβλητή με το όνομα `Sum`
  - Οι **δεσμευμένες λέξεις** της C++ απαγορεύεται να χρησιμοποιηθούν ως ονόματα μεταβλητών

Δεσμευμένες Λέξεις

```
and      const_cast  friend      or_eq        template    volatile
and_eq   continue    goto        private      this        wchar_t
asm      default     if          protected    throw       while
auto     delete      inline      public       true        xor
bitand   do          int         register     try         xor_eq
bitor    double      long        reinterpret_cast
bool     dynamic_cast             return       typedef
break    else        mutable     short        typeid
case     enum        namespace   signed       typename
catch    explicit    new         sizeof       union
char     extern      not         static       unsigned
class    false       not_eq      static_cast  using
compl    float       operator    struct       virtual
const    for         or          switch       void
```

Η C++11 δεσμεύει και τις ακόλουθες λέξεις:

```
alignas alignof  constexpr  char16_t char32_t decltype  noexcept
nullptr static_assert thread_local
```

Παρατηρήσεις

- Το όνομα που επιλέγετε να δώσετε σε μία μεταβλητή είναι χρήσιμο να περιγράφει όσο το δυνατόν καλύτερα τον σκοπό της μεταβλητής μέσα στο πρόγραμμα, ώστε ο κώδικας να είναι πιο ευανάγνωστος
  - Π.χ. το όνομα μίας μεταβλητής που υπολογίζει το άθροισμα κάποιων αριθμών είναι προτιμότερο να είναι **`sum`** αντί για **`var`**

- Αν το όνομα που επιλέξατε για μία μεταβλητή αποτελείται από δύο ή και περισσότερες λέξεις, μία συνήθης πρακτική είναι να διαχωρίζονται μεταξύ τους με τον χαρακτήρα **`'_'`**, έτσι ώστε να διευκολύνεται η ερμηνεία τους
  - Π.χ. το όνομα μίας μεταβλητής που υπολογίζει τον αριθμό των βιβλίων σε μία βιβλιοθήκη είναι προτιμότερο να είναι **`books_number`** αντί για **`booksnumber`**

- Μην δίνετε παρόμοια ονόματα σε μεταβλητές (π.χ. **`more`** και **`More`**), γιατί είναι πολύ εύκολο να κάνετε λάθος και να χρησιμοποιήσετε την μία μεταβλητή στη θέση της άλλης
- Τα ονόματα απλών μεταβλητών συνηθίζεται να γράφονται με πεζά γράμματα, ενώ τα ονόματα σταθερών και μακροεντολών (τις οποίες θα δούμε στη συνέχεια) με κεφαλαία

Δήλωση Μεταβλητών

- Για να χρησιμοποιήσετε μία μεταβλητή μέσα σε ένα πρόγραμμα πρέπει πρώτα να τη δηλώσετε
- Μία μεταβλητή μπορεί να δηλωθεί σε όποιο σημείο του προγράμματος επιθυμούμε
- Η δήλωση μίας μεταβλητής γίνεται με τον ακόλουθο τρόπο:

```
τύπος_δεδομένων όνομα_μεταβλητής;
```

- Το `όνομα_μεταβλητής` είναι το τυχαίο όνομα που επιλέγει ο προγραμματιστής σύμφωνα με τους κανόνες και τις παρατηρήσεις που είπαμε προηγουμένως
- Ο `τύπος_δεδομένων` είναι ένας από τους τύπους δεδομένων που υποστηρίζει η γλώσσα C++
  - Π.χ. η δεσμευμένη λέξη `int` χρησιμοποιείται για τη δήλωση ακέραιων μεταβλητών, δηλαδή μεταβλητών που μπορούν να έχουν μόνο ακέραιες τιμές ενώ η δεσμευμένη λέξη `float` χρησιμοποιείται για τη δήλωση πραγματικών μεταβλητών, δηλαδή μεταβλητών που μπορούν να έχουν τιμές με κλασματικό μέρος

Τύποι Δεδομένων

Τύπος	Συνηθισμένο μέγεθος (bytes)	Εύρος τιμών (min-max)
bool	1	false/true
char	1	-128 … 127
wchar_t	2	-32.768 … 32.767
short int	2	-32.768 … 32.767
int	4	-2.147.483.648…2.147.483.647
long int	4	-2.147.483.648…2.147.483.647
float	4	Μικρότερη θετική τιμή: $1.17*10^{-38}$ Μεγαλύτερη θετική τιμή: $3.4*10^{38}$
double	8	Μικρότερη θετική τιμή: $2.2*10^{-308}$ Μεγαλύτερη θετική τιμή: $1.8*10^{308}$
long double	12, 16	
unsigned char	1	0 … 255
unsigned short int	2	0 … 65535
unsigned int	4	0 … 4.294.967.295
unsigned long int	4	0 … 4.294.967.295

Η C++11 προσθέτει τους παρακάτω τύπους:

`char16_t`: Χρησιμοποιείται για την αποθήκευση συνόλων χαρακτήρων 16 bit, όπως το UTF-16.
`char32_t`: Χρησιμοποιείται για την αποθήκευση συνόλων χαρακτήρων 32 bit, όπως το UTF-32.
`long long int`: Χρησιμοποιείται για πολύ μεγάλους ακεραίους (π.χ. τουλάχιστον 64 bit). Ισχύει `sizeof(long) <= sizeof(long long)`.

Δημιουργία Συνωνύμων Τύπων (1)

- Υπάρχουν πολλές περιπτώσεις όπου η δημιουργία συνωνύμων για τύπους μπορεί να κάνει το πρόγραμμά μας πιο ευέλικτο και πιο εύκολο να διαβαστεί
- Για να δημιουργήσουμε συνώνυμο για έναν υπάρχοντα τύπο δεδομένων μπορούμε να χρησιμοποιήσουμε την δεσμευμένη λέξη `typedef` και με την C++11 την λέξη `using`
- Π.χ., η δήλωση

```
typedef short int short_t;
```

κάνει το όνομα `short_t` συνώνυμο του τύπου `short int`. Άρα, οι δηλώσεις:

`short int i;` και `short_t i;` είναι ισοδύναμες

- Η σύνταξη μοιάζει με τη δήλωση μεταβλητής (π.χ. όπως θα δηλώναμε τη `short_t`), απλά προηγείται η λέξη `typedef`
- Εναλλακτικά, με την `using` γράφουμε `using short_t = short int`
- Προσοχή, η `typedef` (και το ίδιο ισχύει και με την `using`) δεν δημιουργεί μεταβλητή (δηλαδή, το `short_t` δεν είναι μεταβλητή), ούτε ένα νέο τύπο, απλά ένα νέο όνομα

Δημιουργία Συνωνύμων Τύπων (2)

- Η δυνατότητα να δημιουργούμε ένα συνώνυμο για έναν υπάρχοντα τύπο δεδομένων μπορεί να αποβεί πολύ χρήσιμη. Για παράδειγμα, αν η εφαρμογή σας πρόκειται να εκτελεστεί σε συστήματα που οι τύποι δεδομένων εξαρτώνται από το σύστημα, η δημιουργία συνωνύμων παρέχει μεγάλη ευελιξία
- Για παράδειγμα, με την προηγούμενη δήλωση οι μεταβλητές τύπου `short_t` είναι `short int`. Αν όμως σε κάποιο άλλο σύστημα αυτές οι μεταβλητές πρέπει να είναι `int`, απλά αλλάζετε το `short int` σε `int` και οι μεταβλητές γίνονται `int`
- Βλέπουμε λοιπόν ότι η χρήση των συνωνύμων μας βοηθάει να προσαρμόσουμε τους τύπους μας πολύ εύκολα σε συστήματα με διαφορετικές απαιτήσεις
- Συνήθως, η δημιουργία συνωνύμων τοποθετείται σε κάποιο αρχείο επικεφαλίδας, ώστε να μπορούμε να το συμπεριλάβουμε σε όποιο αρχείο κώδικα επιθυμούμε και να χρησιμοποιήσουμε εκεί αυτά τα συνώνυμα

Παρατηρήσεις (1)

- Πολλές μεταβλητές του ίδιου τύπου μπορούν να δηλωθούν στην ίδια γραμμή, αρκεί να διαχωρίζονται μεταξύ τους με κόμμα (,)
  - Δηλαδή, αντί να δηλώσετε τις μεταβλητές `a`, `b` και `c` σε τρεις ξεχωριστές γραμμές:

```
int a;
int b;
int c;
```

    μπορείτε να τις δηλώσετε σε μία γραμμή ως εξής:

```
int a, b, c;
```

- Η δεσμευμένη λέξη `int` μπορεί να παραληφθεί στους ακέραιους τύπους
  - Π.χ., `short` αντί για `short int`

- Οι δεσμευμένες λέξεις μπορούν να βρίσκονται σε οποιαδήποτε σειρά κατά τη δήλωση μιας μεταβλητής
  - Π.χ., `unsigned long int a;`

    είναι το ίδιο με:

    `int long unsigned a;`

Παρατηρήσεις (2)

- Όταν δηλώνεται μία μεταβλητή, ο μεταγλωττιστής δεσμεύει τόσα bytes όσα χρειάζονται για να είναι σε θέση να αποθηκεύσει την τιμή της

- Το μέγεθος της μνήμης που δεσμεύει ένας τύπος δεδομένων μπορεί να διαφέρει από υπολογιστή σε υπολογιστή
  - Δηλαδή, ο τύπος `int` μπορεί να δεσμεύει 8 bytes σε κάποιον υπολογιστή και όχι 4 bytes
  - Για να μάθετε πόσες οκτάδες δεσμεύει ένας τύπος δεδομένων στον υπολογιστή που εργάζεστε πρέπει να χρησιμοποιήσετε τον τελεστή `sizeof` → Θα τον δούμε παρακάτω

- Με τη δήλωση μιας μεταβλητής, ο μεταγλωττιστής γνωρίζει το όνομά της και τη διεύθυνση μνήμης στην οποία βρίσκεται η μεταβλητή αυτή, έτσι, όταν η μεταβλητή χρησιμοποιείται στο πρόγραμμα ο μεταγλωττιστής χρησιμοποιεί το όνομά της και γνωρίζοντας την αντίστοιχη διεύθυνση μνήμης έχει πρόσβαση στο περιεχόμενο της μεταβλητής
- Η C++ υποστηρίζει στατικό έλεγχο τύπων *(statically typed language)* με την έννοια ότι ο έλεγχος της εγκυρότητας της χρήσης των τύπων γίνεται κατά τη μεταγλώττιση και όχι κατά την εκτέλεση του προγράμματος

Παρατηρήσεις (3)

- Να χρησιμοποιείτε τον τύπο `float` μόνο όταν η ακρίβεια των δεκαδικών ψηφίων δεν είναι τόσο σημαντική στο πρόγραμμά σας
- Σε περίπτωση που χρειάζεστε υψηλή ακρίβεια των δεκαδικών ψηφίων, να χρησιμοποιείτε τον τύπο `double`
- Π.χ. ποια πιστεύετε θα είναι η έξοδος του παρακάτω προγράμματος;

```
#include <iostream>
int main()
{
        float a = 3.1;

        if(a == 3.1)
                std::cout << "Yes\n";
        else
                std::cout << "No\n";
        return 0;
}
```

Παρατηρήσεις (3 - Συνέχεια)

Μεγάλη ήττα...!!! Περιμέναμε η έξοδος του προγράμματος να είναι `Yes`, αλλά εμφανίστηκε `No`;

Και ο λόγος οφείλεται στο περιορισμένο μέγεθος του τύπου `float` να αναπαραστήσει ακριβώς τον αριθμό `3.1`, δηλαδή, η τιμή που έχει αποθηκευτεί στο `a` δεν είναι ακριβώς το `3.1`, αλλά μία παραπλήσια τιμή

Να θυμάστε ότι, όταν χρησιμοποιείτε μία πραγματική μεταβλητή σε συγκρίσεις, είναι ασφαλέστερο να την έχετε δηλώσει σαν `double` ή `long double` και όχι `float`. Δηλαδή, αν αντί για `float a;` είχαμε γράψει `double a;` το πρόγραμμα πιθανότατα θα εμφάνιζε `Yes`

Γενικά, όταν ελέγχετε την τιμή μίας πραγματικής μεταβλητής για ισότητα, δεν μπορείτε να είστε απόλυτα σίγουροι για το αποτέλεσμα της σύγκρισης, αφού υπάρχει το ενδεχόμενο ο αριθμός να μην μπορεί να αναπαρασταθεί με την ακρίβεια που υποστηρίζει ο τύπος. Αν μπορεί να αναπαρασταθεί, το αποτέλεσμα της σύγκρισης είναι έγκυρο

Για παράδειγμα, αν αντί με το `3.1` συγκρίνουμε με το `0.5`, η σύγκριση είναι επιτυχής και το πρόγραμμα εμφανίζει `Yes`, επειδή ο αριθμός `0.5` μπορεί να αναπαρασταθεί με την ακρίβεια του τύπου `float` χωρίς να χαθούν δεκαδικά ψηφία

Παρατηρήσεις (3 - Συνέχεια)

Στη γενική περίπτωση λοιπόν, αν πρέπει να συγκρίνετε για ισότητα δύο πραγματικές τιμές `a` και `b` μην γράψετε `if(a == b)`, **δεν** είναι ασφαλές. Ένας σχετικά απλός και σίγουρα πιο ασφαλής τρόπος είναι να ελέγξετε όχι αν οι δύο τιμές είναι ίδιες, αλλά αν η διαφορά τους είναι πολύ μικρή. Για παράδειγμα, μπορούμε να γράψουμε:

```
if(fabs(a-b) <= accuracy)
```

Η `fabs()` είναι συνάρτηση βιβλιοθήκης που δηλώνεται στο cmath και υπολογίζει την απόλυτη τιμή του ορίσματος, δηλαδή, της διαφοράς των `a` και `b`. Για τιμή της `accuracy` να επιλέξετε αυτήν που θεωρείτε ότι προσεγγίζει την ισότητα τους

Εκχώρηση Τιμών σε Μεταβλητές (1)

- Η εκχώρηση μίας τιμής σε μία μεταβλητή γίνεται είτε μαζί με τη δήλωση της μεταβλητής είτε αργότερα

- Π.χ. με την πρώτη εντολή δηλώνεται μία ακέραια μεταβλητή (`int`) με όνομα `a` και μετά της εκχωρείται η τιμή `100`

```
int a;
a = 100;
```

- Εναλλακτικά, θα μπορούσαμε να γράψουμε την εκχώρηση τιμής μαζί με τη δήλωση:

```
int a = 100; ή ακόμα και int a = {100}; ή ακόμα και int a{100};
```

- Επίσης, επιτρέπεται η απόδοση αρχικών τιμών σε περισσότερες από μία μεταβλητές ίδιου τύπου μαζί με τη δήλωσή τους, π.χ.

```
int a = 100, b = 200, c = 300;
```

Εκχώρηση Τιμών σε Μεταβλητές (2)

- Για την εκχώρηση μίας πραγματικής τιμής χρησιμοποιείται η τελεία (.) για το δεκαδικό μέρος και όχι το κόμμα (,) π.χ.

```
float a = 1.24;
```

- Αν μπροστά από μία ακέραια τιμή υπάρχει το ψηφίο 0, τότε αυτή η τιμή ερμηνεύεται σαν οκταδικός αριθμός
  - Π.χ. με την παρακάτω εντολή η τιμή που εκχωρείται στη μεταβλητή a δεν είναι 100, αλλά 64

```
int a = 0100;
```

- Παρομοίως, αν μπροστά από μία ακέραια τιμή υπάρχει το 0x ή το 0X, τότε αυτή η τιμή ερμηνεύεται σαν δεκαεξαδικός αριθμός
  - Π.χ. με την παρακάτω εντολή η τιμή της μεταβλητής a γίνεται 16.

```
int a = 0x10;
```

- Παρομοίως, αν μπροστά από μία ακέραια τιμή υπάρχει το 0b ή το 0B, τότε αυτή η τιμή ερμηνεύεται σαν δυαδικός αριθμός
  - Π.χ. με την παρακάτω εντολή η τιμή της μεταβλητής a γίνεται 15.

```
int a = 0b1111;
```

Παρατηρήσεις (1)

- Η τιμή που εκχωρείται σε μία μεταβλητή πρέπει να συμβαδίζει με τον τύπο της μεταβλητής
  - Π.χ. με την εντολή:

    ```
    int a = 10.9;
    ```

    η τιμή της μεταβλητής a γίνεται 10, γιατί η μεταβλητή a δηλώνεται σαν ακέραια μεταβλητή και όχι σαν πραγματική και το δεκαδικό μέρος αποκόπτεται (Προσοχή!! Δεν στρογγυλοποιείται)

- Η τιμή που εκχωρείται σε μία μεταβλητή πρέπει να είναι μέσα στο επιτρεπτό εύρος τιμών
  - Π.χ. με την εντολή:

    ```
    char ch = 130;
    ```

    η τιμή της μεταβλητής ch δεν γίνεται 130, γιατί το εύρος τιμών μίας μεταβλητής τύπου char είναι από -128 έως 127. Άρα, η τιμή 130 είναι μία τιμή εκτός των επιτρεπτών ορίων

Παρατηρήσεις (2)

- Η τιμή μίας πραγματικής μεταβλητής μπορεί να γραφεί και με επιστημονική σημειογραφία (συνήθως χρησιμοποιείται όταν η τιμή είναι πολύ μικρή ή πολύ μεγάλη)
  - Π.χ. αντί για

```
a = 0.085;
```

  μπορούμε να γράψουμε

```
a = 85E-3;
```

- Το γράμμα **E** ή **e** αναπαριστά το 10, ενώ ο αριθμός που το ακολουθεί είναι η θετική ή αρνητική δύναμη του 10.

- Δηλαδή, η έκφραση 85E-3 αντιστοιχεί στον αριθμό $85*10^{-3}$

Σταθερές (1)

- Σταθερά ονομάζεται μία μεταβλητή που η τιμή της δεν μπορεί να αλλάξει μέσα στο πρόγραμμα
- Για μεγαλύτερη ευελιξία, μία καλή πρακτική είναι η χρήση σταθερών στη θέση τιμών που εμφανίζονται πολλές φορές στο πρόγραμμα. Αν στο μέλλον χρειαστεί να αλλάξουμε την τιμή της σταθεράς την αλλάζουμε σε ένα μόνο σημείο
- Για να δηλωθεί μία μεταβλητή σαν σταθερά, χρησιμοποιούμε τη λέξη `const`
- Μία `const` μεταβλητή πρέπει να αρχικοποιηθεί όταν δηλωθεί και η τιμή της δεν επιτρέπεται να αλλάξει στη συνέχεια του προγράμματος
  - Π.χ. με την επόμενη εντολή η ακέραια μεταβλητή `a` δηλώνεται σαν σταθερά και της εκχωρείται (μόνιμα) η τιμή 10

    ```
    const int a = 10;
    ```

    Αν σε κάποιο σημείο του προγράμματος επιχειρήσουμε να της αλλάξουμε τιμή, π.χ. να γράψουμε:

    ```
    a = 100;
    ```

    τότε ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους για μη επιτρεπτή ενέργεια

Σταθερές (2)

- Εναλλακτικός τρόπος για τη δήλωση μίας σταθεράς είναι η χρήση της οδηγίας **#define**, η οποία χρησιμοποιείται για τη δήλωση μακροεντολών
- Συνήθως, μία μακροεντολή αντιστοιχίζει ένα συμβολικό όνομα με κάποια αριθμητική τιμή
- Για τη δήλωση μακροεντολών, η οδηγία **#define** χρησιμοποιείται ως εξής:

```
#define όνομα_μακροεντολής τιμή
```

- Π.χ. η εντολή:

```
#define NUM 100
```

δηλώνει τη μακροεντολή με όνομα **NUM** και τιμή 100

- Η **NUM** μπορεί να χρησιμοποιηθεί οπουδήποτε μέσα στο πρόγραμμα
- Ο μεταγλωττιστής όταν συναντάει τη **NUM** μέσα στο πρόγραμμα την αντικαθιστά με την τιμή 100

Ρητή Μετατροπή Τύπου (type cast)

- Υπάρχουν περιπτώσεις όπου ο προγραμματιστής επιθυμεί να μετατρέψει τον τύπο δεδομένων μίας έκφρασης σε κάποιον άλλο τύπο δεδομένων
- Η γενική μορφή μίας τέτοιας μετατροπής είναι:

```
(τύπος_δεδομένων)έκφραση ή τύπος_δεδομένων(έκφραση)
```

- Π.χ. έστω η δήλωση: `double a, b, c = 1.23;`

  τότε η εντολή: `a = (int)c;`

  προσαρμόζει **προσωρινά** τον τύπο της `c` από `double` σε `int` και η τιμή του `a` γίνεται `1`
- Η προσαρμογή δεν αλλάζει την τιμή του `c`, δηλαδή, παραμένει ίση με 1.23. Λέγοντας **προσωρινά**, εννοούμε ότι στη συνέχεια του προγράμματος ο τύπος της μεταβλητής `c` συνεχίζει να είναι `double`
- Εκτός από τους παραπάνω τρόπους ρητής προσαρμογής, η C++ υποστηρίζει και άλλους τρόπους με τη χρήση ειδικών τελεστών. Για παράδειγμα, μπορούμε να γράψουμε:

  `a = static_cast<int>(c); // double σε int.`

Παραδείγματα

- Ποια είναι η έξοδος των παρακάτω προγραμμάτων;

```
#include <iostream>
int main()
{
      int i = 100;
      i = i+i;
      i = 2*i;
      std::cout << i+i << ' ' << i << '\n';
      return 0;
}
```

```
#include <iostream>
int main()
{
      int k;
      float i = 3.9, j = 1.2;

      k = i + (int)j;
      std::cout << k - (int)((int)i + j) << '\n';
      return 0;
}
```

Παραδείγματα

```
Έξοδος: 800 400
```

```
Έξοδος: 0
```

Έξοδος Δεδομένων με το cout

- Το cout είναι ένα αντικείμενο της κλάσης ostream
- Εξ'ορισμού, το cout συνδέεται με την προκαθορισμένη έξοδο (π.χ. οθόνη)
- Το cout ακολουθείται από τον τελεστή << και την παράσταση που θέλουμε να εμφανιστεί
- Η C++ υποστηρίζει αρκετούς τρόπους για τη μορφοποίηση της πληροφορίας

Ακολουθίες Διαφυγής

- Για την αναπαράσταση μη εκτυπώσιμων χαρακτήρων (π.χ. Enter) και χαρακτήρων με ειδική σημασία (π.χ. "), η C++ παρέχει τις ακολουθίες διαφυγής (escape sequences)
- Μία ακολουθία διαφυγής αποτελείται από τον χαρακτήρα \ και έναν ειδικό χαρακτήρα. Ο παρακάτω πίνακας εμφανίζει κάποιες συνηθισμένες ακολουθίες διαφυγής

Ακολουθία διαφυγής	Σημασία
`\a`	Χρησιμοποιείται για τη δημιουργία ηχητικού σήματος.
`\b`	Χρησιμοποιείται για τη διαγραφή του τελευταίου χαρακτήρα, όπως το πλήκτρο backspace.
`\n`	Χρησιμοποιείται για την αλλαγή γραμμής, όπως το πλήκτρο Enter.
`\r`	Χρησιμοποιείται για την επαναφορά του δρομέα στην αρχή της τρέχουσας γραμμής.
`\t`	Χρησιμοποιείται για τη μετακίνηση του δρομέα σε μία απόσταση ίση με το μήκος του tab, όπως το πλήκτρο tab.
`\\`	Χρησιμοποιείται για την εμφάνιση της ανάστροφης κεκλιμένης (\).
`\"`	Χρησιμοποιείται για την εμφάνιση των διπλών εισαγωγικών (").

Χειριστές

- Ένας βολικός τρόπος για τη μορφοποίηση της πληροφορίας είναι με τη χρήση χειριστών (manipulators). Οι χειριστές δηλώνονται στον std χώρο ονομάτων. Ο παρακάτω πίνακας εμφανίζει κάποιους συνηθισμένους χειριστές

Χειριστής	Λειτουργία
`endl`	Προκαλεί αλλαγή γραμμής. Το `endl` αδειάζει άμεσα την ενδιάμεση μνήμη εξόδου.
`dec`	Οι τιμές διαβάζονται/εμφανίζονται σε δεκαδική μορφή.
`hex`	Οι τιμές διαβάζονται/εμφανίζονται σε δεκαεξαδική μορφή.
`oct`	Οι τιμές διαβάζονται/εμφανίζονται σε οκταδική μορφή.
`left`	Η έξοδος στοιχίζεται αριστερά με την εισαγωγή χαρακτήρων συμπλήρωσης στο τέλος
`right`	Η έξοδος στοιχίζεται δεξιά με την εισαγωγή χαρακτήρων συμπλήρωσης στην αρχή.
`fixed`	Οι δεκαδικές τιμές εμφανίζονται σε μορφή σταθερής υποδιαστολής.
`scientific`	Οι δεκαδικές τιμές εμφανίζονται σε επιστημονική μορφή.
`showbase`	Προσθέτει τη βάση του αριθμητικού συστήματος (0, 0x) πριν από την τιμή.
`showpos`	Προσθέτει + πριν από θετικές τιμές.
`skipws`	Τα λευκά διαστήματα (π.χ. κενά, στηλοθέτες) που μπορεί να προηγούνται των δεδομένων εισόδου αγνοούνται.
`setfill(c)`	Αν η τιμή δε χωράει στο πλάτος του πεδίου, οι υπόλοιπες θέσεις συμπληρώνονται με το χαρακτήρα συμπλήρωσης c.
`setprecision(n)`	Ο ακέραιος n καθορίζει τα ψηφία της ακρίβειας. Η ακρίβεια παραμένει σε ισχύ για τις επόμενες πράξεις εξόδου.
`setw(n)`	Ο ακέραιος n καθορίζει το πλάτος του επόμενου πεδίου. Αν το n είναι μικρότερο από τους χαρακτήρες που απαιτούνται για την εμφάνιση του πεδίου δεν λαμβάνεται υπόψη. Το πλάτος παραμένει σε ισχύ μόνο για το επόμενο πεδίο.
`uppercase`	Εμφανίζει τα e και x με κεφαλαία.

Παράδειγμα (1)

- Ποια είναι η έξοδος του παρακάτω προγράμματος;

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

int main()
{
     int i = 100;
     double j = 100.536;

     cout << i << oct << '\t' << i << hex << '\t' << i << endl;
     cout << scientific << j << '\t' << fixed << j << endl;
     cout << setprecision(2) << j << '\t' << setprecision(0) << '\t' << j
<< endl;
     cout << showbase << uppercase << i << '\t' << 1.4999 << '\t' <<
showpos << dec << i << endl;
     return 0;
}
```

Παράδειγμα (1)

```
100       144       64
1.005360e+002   100.536000
100.54            101
0x64      1         +100
```

Παράδειγμα (2)

- Ποια είναι η έξοδος του παρακάτω προγράμματος;

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

int main()
{
      int i = 200;
      double j = 1.23497654;

      cout << setw(10) << setfill('*') << i << '\n';
      cout << setw(2) << i << ' ' << setw(7) << fixed << j << '\n';
      cout << setw(8) << setprecision(3) << left << j << '\n';
      cout << setw(7) << showpos << setfill('#') << internal << i << '\n';
      return 0;
}
```

Παράδειγμα (2)

```
********200
200 1.234977
1.235****
+200###
```

Είσοδος Δεδομένων με το `cin`

- Το `cin` είναι ένα αντικείμενο της κλάσης `istream`
- Εξ'ορισμού, το `cin` συνδέεται με την προκαθορισμένη είσοδο (π.χ. πληκτρολόγιο)
- Το `cin` ακολουθείται από τον τελεστή `>>` και την μεταβλητή στην οποία θα αποθηκευτεί η τιμή που θα διαβαστεί
- Όπως και με το `cout`, μπορούμε να χρησιμοποιήσουμε στην ίδια `cin` εντολή πολλές φορές τον τελεστή `>>` για διάβασμα τιμών. Για παράδειγμα, μπορούμε να γράψουμε `cin >> i >> j;`

Παράδειγμα

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

int main()
{
      int i;
      double j;

      cout << "Enter number: ";
      cin >> i; // Διάβασμα ακεραίου και αποθήκευσή του στο i.
      cout << "Enter number: ";
      cin >> j;
      cout << i << '\t' << fixed << j << '\n';
      return 0;
}
```

- Όταν εκτελείται η **cin** εντολή, το πρόγραμμα περιμένει τον χρήστη να πληκτρολογήσει μία τιμή. Συνήθως, πριν από την **cin** εντολή υπάρχει μία **cout** εντολή, που υποδεικνύει στον χρήστη τι δεδομένα πρέπει να εισάγει
- Όταν ο χρήστης εισάγει μία τιμή και πατήσει *Enter*, αυτή η τιμή θα αποθηκευτεί στην αντίστοιχη μεταβλητή

Είσοδος Δεδομένων με Συγκεκριμένη Μορφή

- Υπάρχουν περιπτώσεις, όπου μπορεί να ζητηθεί από τον χρήστη να εισάγει τα δεδομένα με μία συγκεκριμένη μορφή. Για παράδειγμα, ένα πρόγραμμα που απαιτεί η εισαγωγή της ημερομηνίας να γίνεται με τη μορφή ημέρα/μήνας/χρόνος

- Ένας τρόπος για να εξάγουμε τις τιμές που μας ενδιαφέρουν και να τις εκχωρήσουμε σε αντίστοιχες μεταβλητές είναι να χρησιμοποιήσουμε μεταβλητή(ές) με σκοπό την εξαγωγή των χαρακτήρων που δεν μας ενδιαφέρουν. Για παράδειγμα, η μεταβλητή **ch** χρησιμοποιείται για την εξαγωγή του **/**

```
char ch;
int d, m, y;
cout << "Enter date in the form d/m/y: ";
cin >> d >> ch >> m >> ch >> y;
```

- Βέβαια, προϋπόθεση για να λειτουργήσει σωστά ο παραπάνω κώδικας είναι ο χρήστης να εισάγει την ημερομηνία με τον ίδιο τρόπο (π.χ. 19/6/2030).