ΤΡΙΤΗ ΔΙΑΛΕΞΗ

Λογικός τύπος (**bool**)

Ο τύπος **bool** είναι κατάλληλος για την αναπαράσταση ποσοτήτων που μπορούν να πάρουν δύο μόνο τιμές (π.χ. ναι/όχι, αληθές/ψευδές,...).

Τιμές

true ή **false**

Δήλωση

```
bool a;
bool b{true};
```

Εκχώρηση

```
a = false;
```

Ισοδυναμία με ακέραιους

Το **true** ισοδυναμεί με 1, το **false** με 0.
Αντίστροφα: μη μηδενικός ακέραιος μετατρέπεται σε **true**, το 0 ισοδυναμεί με **false**. **Παράδειγμα:** Είσοδος/έξοδος ποσοτήτων τύπου **bool**.

Απλή λογική έκφραση

Σύγκριση δύο ποσοτήτων με τους τελεστές:

Τελεστής	**Σύγκριση**	**Τελεστής**	**Σύγκριση**
==	ίσο	!=	άνισο
>	μεγαλύτερο	>=	μεγαλύτερο ή ίσο
<	μικρότερο	<=	μικρότερο ή ίσο

Παραδείγματα

$x > 10$ `y != 3` `k == 5`

Προτεραιότητα

Οι τελεστές σύγκρισης έχουν χαμηλότερη προτεραιότητα από τους αριθμητικούς τελεστές (επομένως, το `k>3+2` κάνει αυτό που αναμένουμε, τη σύγκριση του `k` με το 5).

Σύνθετη λογική έκφραση

Προκύπτει από ένωση απλών συνθηκών με τους τελεστές && (AND) και || (OR), ή αλλαγή της τιμής μίας λογικής ποσότητας με τον τελεστή ! (NOT). Ισοδύναμες λέξεις: **and**, **or** και **not** αντίστοιχα.

Παραδείγματα

- $10 \leq x \leq 20 \Rightarrow$ `10.0 <= x && x <= 20.0`
- k είναι 3 ή 5 $\Rightarrow$ `k == 3 || k == 5`
- `!(j==3)`

Προτεραιότητες τελεστών

Πολύ Υψηλή	Τελεστής !,
Υψηλή	Αριθμητικοί τελεστές,
Μεσαία	Τελεστές σύγκρισης,
Χαμηλή	Τελεστής &&,
Πολύ Χαμηλή	Τελεστής ||.

Για σιγουριά βάζουμε παρενθέσεις!

Εκχώρηση λογικής έκφρασης σε λογική μεταβλητή

Μια απλή ή σύνθετη λογική έκφραση έχει τιμή **true** ή **false** και μπορεί να εκχωρηθεί σε μεταβλητή τύπου **bool**:

```
bool b,c;

b = k != 5;
c = 10.0 <= x && x <= 20.0;
```

Short-circuit evaluation

Οι σύνθετες λογικές εκφράσεις με δύο μέλη υπολογίζονται από αριστερά προς τα δεξιά και σταματά ο υπολογισμός μόλις προσδιοριστεί η τελική τιμή της συνολικής έκφρασης.
Οι εκφράσεις μπορεί να είναι χρονοβόρες ή να έχουν «παρενέργειες».
Όταν δεν υπολογίζονται, αποφεύγουμε τις συνέπειές τους.

Παράδειγμα

```
int a, b{3};
std::cin >> a;
bool c = a > 5 && ++b < a; // c = (a>5) && ((++b)<a);
std::cout << b << '\n';
```

Πόσο είναι το b;

Αν a>5 τότε b=4, αλλιώς b=3.

Εντολή `if` (1/2)

```
if (λογική_έκφραση) {
        ... // τμήμα A
} else {
        ... // τμήμα B
}
```

Αν η λογική_έκφραση είναι αληθής, εκτελούνται οι εντολές στο τμήμα Α αλλιώς εκτελούνται οι εντολές στο τμήμα Β.

Παράδειγμα

Αν η πραγματική ποσότητα `x` είναι μεγαλύτερη από 0 να τυπώσουμε «θετική» αλλιώς να τυπώσουμε «αρνητική ή 0»:

```
if (x > 0.0) {
   std::cout << u8"Θετική";
} else {
   std::cout << u8"Αρνητική ή 0";
}
```

Εντολή `if` (2/2)

Παρατηρήσεις

- Μπορούμε να παραλείψουμε τα άγκιστρα που περιβάλλουν μία εντολή.
- Μπορούμε να παραλείψουμε το τμήμα από το **`else`** και μετά, αν είναι κενό το block B.
- Ως λογική έκφραση μπορούμε να έχουμε ποσότητα τύπου **`bool`**.

Παράδειγμα

Η μεταβλητή `y` να αλλάξει πρόσημο αν η πραγματική ποσότητα `x` είναι μεταξύ -1 και 1:

```
bool a = -1.0 <= x && x <= 1.0;
if (a)
    y = -y;
```

Τριαδικός τελεστής ? : (1/2)

Τελεστής με *τρία* ορίσματα: μια λογική συνθήκη και δυο εκφράσεις οποιουδήποτε είδους.

Σύνταξη

Η έκφραση

λογική_έκφραση ? έκφρασηΑ : έκφρασηΒ

έχει την τιμή «έκφρασηΑ» όταν η «λογική_έκφραση» είναι αληθής, ενώ έχει την τιμή «έκφρασηΒ» όταν η λογική_έκφραση είναι ψευδής. Ο υπολογισμός της κατάλληλης έκφρασης γίνεται μετά την επιλογή της.

Τριαδικός τελεστής ?: (2/2)

Παράδειγμα

Το

```
val = (συνθήκη ? value1 : value2);
```

ισοδυναμεί με

```
if (συνθήκη) {
   val = value1;
} else {
   val = value2;
}
```

Μπορεί να εμφανιστεί και στο *αριστερό μέρος* του τελεστή εκχώρησης (με παρενθέσεις):

```
(k == 5 ? a : b) = 3;
```

Εντολή `switch` (1/3)

```
switch (i) {
  case v1:
  ...
  case v2:
  ...
  case vN:
  ...
  default:
  ...
}
```

Ελέγχεται το i.
Αν είναι ίσο με v1 εκτελούνται όλες οι εντολές μετά το **case** v1.
Αν είναι ίσο με v2 εκτελούνται όλες οι εντολές μετά το **case** v2, κλπ.
Αν δεν είναι ίσο με κανένα, εκτελούνται όλες οι εντολές μετά το **default**.

Παρατηρήσεις

- Το i είναι ακέραιος τύπος (**int** και παραλλαγές, **char**, **bool**).
- τα v1, v2, ... είναι υποχρεωτικά σταθερές τιμές.
- Σχεδόν πάντα είναι απαραίτητο το **break** στο τέλος κάθε block εντολών ώστε να διακόπτεται η εκτέλεση του **switch**.

Εντολή **switch**. Α' Παράδειγμα (2/3)

Αν ο ακέραιος k είναι 1,2,3,4 να τυπώσουμε "up", "down", "left", "right" αντίστοιχα. Για άλλη τιμή να γράψουμε "error":

```
switch (k) {
case 1:
  std::cout << "up\n";
  break;
case 2:
  std::cout << "down\n";
  break;
case 3:
  std::cout << "left\n";
  break;
case 4:
  std::cout << "right\n";
  break;
default:
  std::cout << "error\n";
}
```

Εντολή **switch**. Β' Παράδειγμα (3/3)

Ομαδοποίηση τιμών

Αν ο ακέραιος k είναι 2, 4, 6, 8 να τυπώσουμε "ζυγός". Αν είναι 1, 3, 5, 7, 9 να τυπώσουμε "μονός". Αν είναι αρνητικός να τυπώσουμε "λάθος":

```
switch (k) {
case 2: case 4:
case 6: case 8:
   std::cout << u8"ζυγός\n";
   break;
case 1: case 3:
case 5: case 7: case 9:
   std::cout << u8"μονός\n";
   break;
default:
  std::cout << u8"λάθος\n";
}
```

Μιγαδικός Τύπος (1/3)

Υποστηρίζεται ο μιγαδικός τύπος αφού συμπεριληφθεί ο header `<complex>`. Αν γράψουμε και την εντολή
`using namespace std::complex_literals;`
μπορούμε να γράψουμε φανταστικούς αριθμούς με απλό τρόπο.

Δηλώσεις (με απόδοση αρχικής τιμής)

- `std::complex<double> z{3.4,2.8}; // z = 3.4 + 2.8 i`
- `std::complex<double> z{3.4}; // z = 3.4 + 0.0 i`
- `std::complex<double> z = 3.4; // z = 3.4 + 0.0 i`
- `std::complex<double> z = 2.8i; // z = 0.0 + 2.8 i`
- `std::complex<double> z{}; // z = 0.0 + 0.0 i`
- `std::complex<double> z; // z = 0.0 + 0.0 i`
- `std::complex<double> w{z}; // w = z`

Μιγαδικός Τύπος (2/3)

Πράξεις και συναρτήσεις μιγαδικών ποσοτήτων

- Οι αριθμητικοί τελεστές '+', '−', '∗', '/' και οι συντμήσεις '+=', '−=', '∗=', '/=' μεταξύ μιγαδικών αριθμών ίδιου τύπου ή ενός μιγαδικού και ενός πραγματικού αριθμού με ίδιο βασικό τύπο εκτελούν τις αναμενόμενες και γνωστές πράξεις από τα μαθηματικά και έχουν μιγαδικό αποτέλεσμα.
- Οι μαθηματικές συναρτήσεις που έχουν νόημα για μιγαδικούς αριθμούς, δέχονται μιγαδικά ορίσματα και επιστρέφουν το αντίστοιχο μιγαδικό αποτέλεσμα (εκτός από τη συνάρτηση `std::abs()`).

Μιγαδικός Τύπος (3/3)

Επιπλέον συναρτήσεις για μιγαδικούς

- `std::abs(z)` $\rightarrow \sqrt{\mathtt{z}\mathtt{z}^\star}$.
- `std::polar(r,t)` $\rightarrow r\mathrm{e}^{\mathrm{i}t}$. Αν παραληφθεί το t θεωρείται 0.0.
- `std::norm(z)` $\rightarrow \mathtt{z}\mathtt{z}^\star$.
- `std::arg(z)` $\rightarrow \tan^{-1}(\beta/\alpha)$, αν $z = \alpha + \mathrm{i}\beta$.
- `std::conj(z)` $\rightarrow \mathtt{z}^\star$.
- `std::real(z)` $\rightarrow \alpha$, αν $z = \alpha + \mathrm{i}\beta$.
- `std::imag(z)` $\rightarrow \beta$, αν $z = \alpha + \mathrm{i}\beta$.

Για να αποδώσουμε νέα τιμή στο πραγματικό ή φανταστικό μέρος μιας μιγαδικής μεταβλητής z, μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις–μέλη `real()` και `imag()` με ένα όρισμα, τη νέα τιμή:

```
std::complex<double> z{3.0,1.5}; // z = 3.0 + 1.5i
z.real(4.2);  // z = 4.2 + 1.5i
z.imag(-0.9); // z = 4.2 - 0.9i
```

Είσοδος/Έξοδος μιγαδικών δεδομένων (1/2)

Η εκτύπωση μιγαδικών δεδομένων με τον τελεστή ‘<<’ γίνεται με τη μορφή

(πραγματικό μέρος,φανταστικό μέρος)

Η ανάγνωση μιγαδικών δεδομένων με τον τελεστή ‘>>’ γίνεται με μια από τις παρακάτω διαμορφώσεις:

(πραγματικό μέρος,φανταστικό μέρος)
(πραγματικό μέρος)
πραγματικό μέρος

Είσοδος/Εξοδος μιγαδικών δεδομένων (2/2)

Παράδειγμα

Οι εντολές

```
std::complex<double> z{3.2,1.5};
std::cout << z;
```

εμφανίζουν στην οθόνη

```
(3.2,1.5)
```

Οι εντολές

```
std::complex<double> w;
std::cin >> w;
```

περιμένουν κάτι σαν

```
(2.3, 1.5) ή (2.3) ή 2.3
```

Εντολή `using`

Μπορούμε να ορίσουμε μια άλλη, ισοδύναμη αλλά ίσως πιο σύντομη, *ονομασία* για τύπο με τη βοήθεια της εντολής **`using`**.

Σύνταξη

`using` *νέο_όνομα_τύπου = παλαιό_όνομα_τύπου;*

Δεν δημιουργείται νέος τύπος αλλά αποκτά νέο όνομα ο αρχικός.

Παράδειγμα

```
using complex = std::complex<double>;

std::complex<double> z{3.4,1.4};
complex w{2.8,-1.2};
```

Αναφορά (1/2)

Μπορούμε να ορίσουμε μια άλλη, ισοδύναμη αλλά ίσως πιο σύντομη, *ονομασία* για ποσότητα (μεταβλητή, σταθερή, συνάρτηση, κλπ.):

Σύνταξη

τύπος όνομαA;
τύπος & όνομαB{όνομαA};
auto *& όνομαB = όνομαA;*

Δεν δημιουργείται νέα ποσότητα αλλά μόνο αποκτά νέο όνομα η αρχική.

Παράδειγμα

```
int a{3};
auto b = a; // b = 3
auto & c = a;
b=5; // a = 3
c=7; // a = 7
```

Αναφορά (2/2)

Αν η αρχική ποσότητα είναι σταθερή (**const** ή **constexpr**) πρέπει και η αναφορά να είναι **const**:

Παράδειγμα

```
int constexpr a{5};
int & p{a};   // Λάθος
int const & q{a};
/* Σωστό. Δεν μπορεί να αλλάξει το a μέσω του q */
```

Μπορούμε να ορίσουμε (σταθερή) αναφορά σε σταθερή, μεταβλητή ή έκφραση κατάλληλου τύπου:

Παράδειγμα

```
int const & p{4};
int a{3};
int const & q{a};
int const & r{2*a};
```

Χώρος ονομάτων (1/2)

Μπορούμε να οργανώσουμε τις μεταβλητές, σταθερές, συναρτήσεις, κλάσεις, κλπ. σε χώρους ονομάτων (**namespace**s):

```
namespace my {
int a;
}
namespace your {
double a;
}
```

Οι δύο μεταβλητές με το ίδιο όνομα είναι διαφορετικές. Εκτός των δύο χώρων ονομάτων, η πρώτη ονομάζεται `my::a` και η δεύτερη `your::a`. Εντός κάθε χώρου η "τοπική" μεταβλητή λέγεται απλώς `a`.

Χώρος ονομάτων (2/2)

- Όλες οι ποσότητες που παρέχονται από τη Standard Library ορίζονται στο χώρο ονομάτων `std`. Δεν μπορούμε να προσθέσουμε τίποτε σε αυτόν.
- Αν χρειαστεί να χρησιμοποιήσουμε πολλές φορές, ποσότητες από ένα χώρο ονομάτων π.χ. το std, μπορούμε να δώσουμε την εντολή

  ```
  using namespace std;
  ```

  στο block που περικλείει τον κώδικά μας. Μετά από αυτή, (όλα) τα ονόματα των μεταβλητών, συναρτήσεων κλπ. δεν χρειάζονται το `std::`.
- Αν χρησιμοποιούμε πολλές φορές συγκεκριμένη ποσότητα από ένα χώρο ονομάτων, μπορούμε να παραλείπουμε το όνομα του **namespace** για αυτή μετά από εντολή σαν κι αυτή:

  ```
  using std::cout;
  ```

Ανώνυμος χώρος ονομάτων

Χρήσιμος είναι ο ανώνυμος χώρος ονομάτων:

```
namespace {
    ...
}
```

Οι ποσότητες που ορίζονται σε αυτόν χρησιμοποιούνται απευθείας με το όνομά τους μόνο στο αρχείο που ορίζεται ο ανώνυμος **namespace** (και σε όσα το συμπεριλαμβάνουν με **#include**). Δεν μπορούν να χρησιμοποιηθούν σε άλλο αρχείο και, επομένως, να «συγκρουστούν» με ποσότητες που ορίζονται εκεί.

Ασκήσεις

Δεύτερο κεφάλαιο

20

Τρίτο κεφάλαιο

1,2,3,4,6,7,8,9,10,11