# Αριθμητικοί Τελεστές στη C++

Η συνάρτηση που ακολουθεί δείχνει τη βασική χρήση των αριθμητικών τελεστών στη C++:

void arithmetic() {  
 int a = 10, b = 12;  
 cout << "a+b = " << (a+b)  
 << endl  
 << "a-b = " << (a-b)  
 << endl  
 << "a\*b = " << (a\*b)  
 << endl;  
}

## 1. Θεωρία

Οι αριθμητικοί τελεστές χρησιμοποιούνται για να εκτελέσουν βασικές μαθηματικές πράξεις σε αριθμητικούς τύπους όπως int, float, double κ.λπ.

|  |  |  |  |
| --- | --- | --- | --- |
| Τελεστής | Περιγραφή | Παράδειγμα | Αποτέλεσμα |
| + | Πρόσθεση | 10 + 12 | 22 |
| - | Αφαίρεση | 10 - 12 | -2 |
| \* | Πολλαπλασιασμός | 10 \* 12 | 120 |
| / | Διαίρεση | 10 / 2 | 5 |
| % | Υπόλοιπο διαίρεσης (modulo) | 10 % 3 | 1 |

## 2. Ανάλυση Παραδείγματος

Στη συνάρτηση, δηλώνονται δύο ακέραιες μεταβλητές a και b με τιμές 10 και 12. Στη συνέχεια, εκτελούνται πράξεις πρόσθεσης, αφαίρεσης και πολλαπλασιασμού και τα αποτελέσματα εμφανίζονται στην οθόνη μέσω της cout.

Παράδειγμα εξόδου:

a+b = 22  
a-b = -2  
a\*b = 120

## 3. Προτεραιότητα και Σειρά Εκτέλεσης

Οι αριθμητικοί τελεστές έχουν προκαθορισμένη σειρά προτεραιότητας (precedence):

|  |  |  |
| --- | --- | --- |
| Επίπεδο Προτεραιότητας | Τελεστές | Κατεύθυνση |
| Υψηλότερη | \*, /, % | Αριστερά προς δεξιά |
| Χαμηλότερη | +, - | Αριστερά προς δεξιά |

Σε έκφραση όπως a + b \* c, η C++ εκτελεί πρώτα το b \* c και μετά προσθέτει το a. Η σειρά μπορεί να τροποποιηθεί με χρήση παρενθέσεων.

## 4. Τύποι και Αριθμητική

Όταν και οι δύο τελεστέοι είναι ακέραιοι, η πράξη γίνεται με ακέραια αριθμητική. Αν τουλάχιστον ένας είναι δεκαδικός (float ή double), η πράξη γίνεται με δεκαδική αριθμητική.

int x = 5, y = 2;  
cout << x / y; // 2 (ακέραιη διαίρεση)  
cout << (float)x / y; // 2.5 (δεκαδική διαίρεση)

# Ακέραιη και Πραγματική Διαίρεση στη C++

Η ακόλουθη συνάρτηση παρουσιάζει τη διαφορά ανάμεσα στην ακέραιη και την πραγματική (δεκαδική) διαίρεση στη C++:

void division() {  
 int a=3, b=2;  
 cout << "Integer Division: " << a << '/' << b << " = " << (a/b) << endl;  
 cout << "Real Division: " << a << '/' << b << " = " << ((float)a/b) << endl;  
  
 b=0;  
 cout << "Real Division: " << a << '/' << b << " = " << ((float)a/b) << endl;  
 cout << "Real Division: " << a << '/' << b << " = " << (a/b) << endl;  
}

## 1. Θεωρία

Στη C++ ο τελεστής διαίρεσης (/) μπορεί να λειτουργήσει με δύο διαφορετικούς τρόπους, ανάλογα με τους τύπους των τελεστέων:

|  |  |  |
| --- | --- | --- |
| Τύπος Τελεστέων | Είδος Διαίρεσης | Αποτέλεσμα |
| int / int | Ακέραιη διαίρεση | Το αποτέλεσμα είναι ακέραιος, τα δεκαδικά απορρίπτονται |
| float / int | Πραγματική διαίρεση | Το αποτέλεσμα είναι δεκαδικός αριθμός (float ή double) |
| int / float | Πραγματική διαίρεση | Το αποτέλεσμα είναι δεκαδικός αριθμός |

## 2. Παράδειγμα Εξόδου

Αν εκτελέσουμε τη συνάρτηση, θα πάρουμε την εξής έξοδο:

Integer Division: 3/2 = 1  
Real Division: 3/2 = 1.5

## 3. Ανάλυση

Στην πρώτη περίπτωση, τόσο το a όσο και το b είναι ακέραιοι (int). Συνεπώς, η πράξη 3/2 εκτελείται ως \*\*ακέραιη διαίρεση\*\*, και το αποτέλεσμα είναι 1, καθώς τα δεκαδικά απορρίπτονται (η C++ δεν στρογγυλοποιεί).

Στη δεύτερη περίπτωση, το a μετατρέπεται σε float με την έκφραση (float)a. Έτσι, η διαίρεση γίνεται \*\*πραγματική (δεκαδική)\*\* και το αποτέλεσμα είναι 1.5.

## 4. Διαίρεση με Μηδέν

Η C++ αντιμετωπίζει διαφορετικά τη διαίρεση με το μηδέν ανάλογα με τον τύπο των τελεστέων:

|  |  |
| --- | --- |
| Περίπτωση | Συμπεριφορά |
| Ακέραιη διαίρεση (int / 0) | Μη καθορισμένη συμπεριφορά – μπορεί να προκαλέσει σφάλμα χρόνου εκτέλεσης |
| Πραγματική διαίρεση (float / 0) | Το αποτέλεσμα είναι ±∞ (άπειρο) σύμφωνα με το πρότυπο IEEE-754 |

## 5. Συμπέρασμα

Η ακέραιη διαίρεση επιστρέφει μόνο το ακέραιο μέρος του αποτελέσματος, απορρίπτοντας τα δεκαδικά, ενώ η πραγματική διαίρεση υπολογίζει το πλήρες δεκαδικό αποτέλεσμα. Η μετατροπή ενός από τους τελεστέους σε float ή double μετατρέπει ολόκληρη την πράξη σε πραγματική διαίρεση.

# Ο Τελεστής Modulo (%) στη C++

Η ακόλουθη συνάρτηση παρουσιάζει τη χρήση του τελεστή υπολοίπου διαίρεσης (%) στη C++:

void modulo() {  
 int a=3, b=2;  
 cout << "Integer Modulo: " << a << '%' << b << " = " << (a%b) << endl;  
  
 b=0;  
 cout << "Integer Modulo: " << a << '%' << b << " = " << (a%b) << endl;  
}

## 1. Θεωρία

Ο τελεστής modulo (%) υπολογίζει το υπόλοιπο της ακέραιης διαίρεσης δύο αριθμών. Επιστρέφει το μέρος της διαίρεσης που απομένει αφού αφαιρεθούν όλα τα πολλαπλάσια του διαιρέτη (b) από τον διαιρετέο (a). Χρησιμοποιείται αποκλειστικά για ακέραιους τύπους (int, long, short, char).

|  |  |  |  |
| --- | --- | --- | --- |
| Έκφραση | Πράξη | Αποτέλεσμα Διαίρεσης | Υπόλοιπο (%) |
| 5 % 2 | 5 / 2 = 2 | Πηλίκο = 2 | Υπόλοιπο = 1 |
| 10 % 3 | 10 / 3 = 3 | Πηλίκο = 3 | Υπόλοιπο = 1 |
| 14 % 4 | 14 / 4 = 3 | Πηλίκο = 3 | Υπόλοιπο = 2 |
| 7 % 7 | 7 / 7 = 1 | Πηλίκο = 1 | Υπόλοιπο = 0 |

## 2. Παράδειγμα Εξόδου

Αν εκτελέσουμε τη συνάρτηση, το αποτέλεσμα θα είναι:

Integer Modulo: 3%2 = 1

## 3. Ανάλυση

Η πράξη 3 % 2 επιστρέφει 1, γιατί το 3 διαιρείται με το 2 μία φορά (3 / 2 = 1) και απομένει υπόλοιπο 1.

Ο τελεστής % εφαρμόζεται μόνο σε \*\*ακέραιους τύπους\*\*. Αν προσπαθήσουμε να τον χρησιμοποιήσουμε σε float ή double, ο μεταγλωττιστής θα δώσει σφάλμα. Για δεκαδικούς αριθμούς χρησιμοποιούμε τη συνάρτηση fmod() από τη βιβλιοθήκη <cmath>.

#include <cmath>  
float result = fmod(5.5, 2.0); // αποτέλεσμα: 1.5

## 4. Modulo με Αρνητικούς Αριθμούς

Στη C++ το πρόσημο του αποτελέσματος της % ακολουθεί το πρόσημο του πρώτου τελεστέου (του διαιρετέου). Παραδείγματα:

5 % 2 = 1  
-5 % 2 = -1  
5 % -2 = 1  
-5 % -2 = -1

## 5. Modulo με Μηδέν

Η έκφραση a % 0 έχει \*\*μη καθορισμένη συμπεριφορά\*\* (undefined behavior). Δεν επιτρέπεται να γίνει modulo ή διαίρεση με το μηδέν και το πρόγραμμα συνήθως τερματίζει με σφάλμα χρόνου εκτέλεσης.

## 6. Συμπέρασμα

Ο τελεστής % υπολογίζει το υπόλοιπο της ακέραιης διαίρεσης. Χρησιμοποιείται μόνο με ακέραιους τύπους και δεν μπορεί να εφαρμοστεί σε δεκαδικούς. Για αριθμούς κινητής υποδιαστολής χρησιμοποιούμε τη fmod(). Η κατανόηση του τελεστή αυτού είναι σημαντική για επαναληπτικούς υπολογισμούς, αλγορίθμους υπολοίπου, έλεγχο ζυγών/μονών αριθμών και δημιουργία κυκλικών ακολουθιών.

# Τελεστές Αυξομείωσης (++i και i++) στη C++

Η ακόλουθη συνάρτηση παρουσιάζει τη διαφορά μεταξύ των τελεστών προ-αύξησης (pre-increment) και μετα-αύξησης (post-increment) στη C++:

void increment() {  
 int i=0;  
 cout << "i++ = " << i++ << endl; // post-increment  
 cout << "i = " << i << endl;  
  
 i=0;  
 cout << "++i = " << ++i << endl; // pre-increment  
 cout << "i = " << i << endl;  
 // Τα ίδια ισχύουν και για τους τελεστές μείωσης i-- και --i  
}

## 1. Θεωρία

Οι τελεστές ++ και -- χρησιμοποιούνται για την αύξηση ή μείωση μιας μεταβλητής κατά 1. Υπάρχουν δύο τρόποι χρήσης τους:

|  |  |  |
| --- | --- | --- |
| Μορφή | Όνομα | Περιγραφή |
| i++ | Post-increment | Αυξάνει τη μεταβλητή κατά 1, αλλά επιστρέφει την αρχική τιμή πριν την αύξηση |
| ++i | Pre-increment | Αυξάνει τη μεταβλητή κατά 1 και επιστρέφει τη νέα τιμή μετά την αύξηση |
| i-- | Post-decrement | Μειώνει τη μεταβλητή κατά 1, αλλά επιστρέφει την αρχική τιμή |
| --i | Pre-decrement | Μειώνει τη μεταβλητή κατά 1 και επιστρέφει τη νέα τιμή |

## 2. Ανάλυση Παραδείγματος

Αρχικά, η μεταβλητή i έχει την τιμή 0.

Η εντολή:

cout << "i++ = " << i++ << endl;

Εκτελείται ως εξής: η τιμή του i (0) χρησιμοποιείται στην έκφραση και στη συνέχεια το i αυξάνεται κατά 1. Άρα εμφανίζεται πρώτα 0 και μετά το i γίνεται 1.

Στη συνέχεια, μετά την εντολή:

cout << "i = " << i << endl;

Εμφανίζεται η νέα τιμή του i (1).

Στο δεύτερο μέρος του παραδείγματος, το i αρχικοποιείται πάλι σε 0.

Η εντολή:

cout << "++i = " << ++i << endl;

Αυτή τη φορά, η αύξηση γίνεται πρώτα και μετά η νέα τιμή (1) χρησιμοποιείται στην έκφραση. Έτσι, εμφανίζεται αμέσως το 1.

Τελικά η εντολή:

cout << "i = " << i << endl;

Εμφανίζει πάλι 1, καθώς το i έχει ήδη αυξηθεί.

## 3. Παράδειγμα Εξόδου

i++ = 0  
i = 1  
++i = 1  
i = 1

## 4. Διαφορά στη Χρήση

Η διαφορά μεταξύ των δύο τελεστών γίνεται εμφανής όταν χρησιμοποιούνται σε πιο σύνθετες εκφράσεις. Για παράδειγμα:

int i = 5;  
int a = i++; // a=5, i=6 (post-increment)  
int b = ++i; // i=7, b=7 (pre-increment)

Η επιλογή του κατάλληλου τελεστή εξαρτάται από το πότε θέλουμε να αυξηθεί η μεταβλητή — πριν ή μετά τη χρήση της τιμής της.

## 5. Συμπέρασμα

Οι τελεστές προ- και μετα- αύξησης (++i, i++) είναι σύντομοι τρόποι για να αυξάνουμε μια μεταβλητή κατά 1. Η βασική τους διαφορά είναι το πότε εφαρμόζεται η αύξηση σε σχέση με τη χρήση της μεταβλητής στην έκφραση. Οι ίδιες αρχές ισχύουν και για τους τελεστές μείωσης (--i, i--).

# Σύγκριση Πραγματικών Αριθμών στη C++ (==)

Η ακόλουθη συνάρτηση δείχνει γιατί η απευθείας σύγκριση δύο πραγματικών αριθμών (float ή double) με τον τελεστή ισότητας (==) μπορεί να αποδώσει λανθασμένα αποτελέσματα λόγω σφαλμάτων στρογγυλοποίησης:

void realsEquality() {  
 float f1,f2;  
 float t=0.000001;  
  
 f1=t\*10;  
 f2=t+t+t+t+t+t+t+t+t+t;  
 cout << setprecision(10);  
 cout << f1 <<" "<<f2<<" "<< (f1==f2);  
}

## 1. Παρατήρηση

Στο παραπάνω πρόγραμμα, οι δύο εκφράσεις f1 και f2 φαίνεται να υπολογίζουν την ίδια μαθηματική τιμή (10\*0.00001). Ωστόσο, η εντολή (f1 == f2) επιστρέφει false (0).

## 2. Γιατί Συμβαίνει Αυτό

Οι πραγματικοί τύποι float και double δεν μπορούν να αναπαριστήσουν άπειρο πλήθος τιμών. Ωστόσο οι πραγματικοί αριθμοί είναι άπειροι. Έστω οι μεταβλητές d1 kai d2, τύπου double και d1-d2 είναι ο ελάχιστος πραγματικός του τύπου double, ας τον ονομάσουμε minDouble. Επομένως, όταν στους double αναπαρίσταται ο d1, ο αμέσως επόμενος που μπορεί να αναπαρασταθεί είναι ο d2=d1+minDouble, μετά ο d2+minDouble, κοκ. Ωστόσο, μεταξύ d1 και d2 υπάρχουν άπειρες πραγματικές τιμές. Καμία όμως δεν μπορεί να αναπαρασταθεί. Κατά συνέπεια, όταν το αποτέλεσμα ενός υπολογισμού είναι ένας αριθμός μεταξύ d1 και d2, αυτό θα αποθηκευθεί ως d1 ή d2. Έχουμε δηλαδή απώλεια ακρίβειας.

## 3. Εμφάνιση με Περισσότερη Ακρίβεια

Η εντολή cout << setprecision(10) δείχνει περισσότερα δεκαδικά ψηφία, επιτρέποντάς μας να δούμε τη μικρή διαφορά που κανονικά κρύβεται από τη cout.

## 4. Σύγκριση Πραγματικών Αριθμών

Η σωστή πρακτική είναι να μην χρησιμοποιούμε τον τελεστή ισότητας (==) απευθείας για float ή double, αλλά να ελέγχουμε αν η απόλυτη διαφορά τους είναι μικρότερη από ένα πολύ μικρό όριο (επίσης γνωστό ως epsilon).

if (fabs(f1 - f2) < 1e-7)  
 cout << "Σχεδόν ίσα" << endl;  
else  
 cout << "Διαφέρουν" << endl;

## 5. Επιλογή του Εύρους (Epsilon)

Η τιμή του ορίου (epsilon) εξαρτάται από το μέγεθος των αριθμών και από τον τύπο (float ή double). Για float συνήθως χρησιμοποιείται 1e-6 έως 1e-7, ενώ για double 1e-10 έως 1e-12.

## 6. Συμπέρασμα

Ο τελεστής == δεν πρέπει να χρησιμοποιείται απευθείας για σύγκριση πραγματικών αριθμών, διότι οι μικρές αποκλίσεις λόγω στρογγυλοποίησης μπορούν να κάνουν δύο τιμές που θεωρητικά είναι ίσες να φαίνονται άνισες. Αντί γι’ αυτό, συγκρίνουμε τη διαφορά τους με ένα μικρό επιτρεπτό όριο (epsilon).