



Πανεπιστήμιο Αιγαίου

Μεθοδολογίες και Γλώσσες Προγραμματισμού I

Εξαιρέσεις (Exceptions)

Εργίνα Καβαλλιεράτου (kavallieratou@aegean.gr)

Μόνιμη Επίκουρος Καθηγήτρια

Τμήμα Μηχανικών Πληροφοριακών & Επικοινωνιακών Συστημάτων



Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο



ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης



Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο



ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης



Λάθη

- ✓ Δεν μπορείτε να προβλέψετε όλες τις εξαιρετικές περιστάσεις, λάθη.
- ✓ Μπορείτε μόνο να προετοιμαστείτε.
- ✓ Οι επιλογές σας περιορίζονται σε:
 - συντριβή του προγράμματος (crash)
 - ενημέρωση του χρήστη και έξοδος με χάρη.
 - ενημέρωση του χρήστη και να προσπαθήσει να ανακτήσει και να συνεχίσει.
 - λήψη διορθωτικών μέτρων και χωρίς να διαταραχθεί ο χρήστης.
- ✓ Η C ++ παρέχει ολοκληρωμένη μέθοδο για την αντιμετώπιση ασυνήθιστων συνθηκών που προκύπτουν κατά την εκτέλεση ενός προγράμματος.

Code rot

- ✓ Ο όρος Σήψη κώδικα, χρησιμοποιείται για να εξηγήσει πώς bug-free κώδικας ξαφνικά γίνεται αναξιόπιστος.
- ✓ Τα προγράμματα είναι εξαιρετικά πολύπλοκα, και σφάλματα και λάθη μπορούν να κρυφτούν για πολύ.
- ✓ Προστατέψτε τον εαυτό σας με το γράψιμο κώδικα εύκολο στη συντήρηση.
- ✓ Αυτό σημαίνει ότι ο κωδικός σας θα πρέπει να σχολιαστεί ακόμα και αν δεν περιμένουμε κανέναν άλλο ποτέ να το δει.
- ✓ Έξι μήνες μετά, τον κωδικό σας θα το διαβάσετε με τα μάτια ενός ξένου.

Exceptions

Οι εξαιρέσεις είναι σφάλματα τα οποία μπορούν να παρουσιαστούν κατά το χρόνο εκτέλεσης (run time) ενός προγράμματος. Τα σφάλματα αυτά αφορούν σε καταστάσεις τις οποίες δεν μπορεί να χειριστεί το πρόγραμμα μας και συνήθως ανήκουν στις κατηγορίες:

- Διαίρεση με το μηδέν
- Εξάντληση της ελεύθερης μνήμης στο σύστημα
- Προβλήματα στο άνοιγμα, στην εγγραφή ή στην ανάγνωση αρχείων
- Πρόσβαση εκτός ορίων στον πίνακα

Exceptions

- ✓ Οι εξαιρέσεις δεν ανήκουν στη φυσιολογική λειτουργία ενός προγράμματος και χρειάζονται άμεση αντιμετώπιση ώστε το πρόγραμμα να τις αντιμετωπίσει με επιτυχία, ή απλά να ενημερώσει το χρήστη για το συγκεκριμένο πρόβλημα
- ✓ Η C++ διαθέτει ένα ενσωματωμένο υποσύστημα χειρισμού εξαιρέσεων, το οποίο επιτρέπει τον χειρισμό των εξαιρέσεων με ένα αυτοματοποιημένο, δομημένο και ελεγχόμενο τρόπο.
- ✓ Αυτό το χαρακτηριστικό της γλώσσας επιτρέπει τα διάφορα τμήματα του προγράμματος να επικοινωνούν με ένα ενιαίο σύστημα διαχείρισης σφαλμάτων, το οποίο είναι ανεξάρτητο (πολλές φορές αναπτυγμένο ξεχωριστά) από το κυρίως πρόγραμμα.

Exception

- ✓ Δοκιμάζοντας την κανονική ροή: Έλεγχος της αναμενόμενης συμπεριφορά ενός προγράμματος, με το `try`:

`try` {Συμπεριφορά}

Το `try` απαιτείται. Γνωστοποιεί στο compiler ότι θα πρόβλεψετε μια μη φυσιολογική συμπεριφορά και θα προσπαθήσει να ασχοληθεί με το θέμα. Η πραγματική συμπεριφορά που πρέπει να αξιολογηθεί περιλαμβάνεται μεταξύ αγκίστρων.

Εκεί περιλαμβάνεται η κανονική ροή που θα πρέπει να ακολουθήσει το πρόγραμμα.

Exception

- ✓ **Catching Λάθη:** σε περίπτωση μη φυσιολογικής συμπεριφοράς, μπορείτε να μεταφέρετε τη ροή του προγράμματος σε ένα άλλο τμήμα που μπορεί να ασχοληθεί με το θέμα. Η σύνταξη είναι:

catch(Παράμετρος) {Τι να κάνει}

- ✓ Το **catch** απαιτείται και ακολουθεί το **try**. Χρησιμοποιεί παράμετρο από το **try**. Μπορεί να είναι μια μεταβλητή ή μια κλάση. Εάν δεν συντάσσεται ως **catch(...)**.

Σύνταξη

```
try
{
    // Έλεγχος ροής
}
catch(Παράμετρος)
{
    // Catch the exception
}
```

Exception

- ✓ **Throwing** ένα σφάλμα: για μια μη φυσιολογική συμπεριφορά του προγράμματος μεταφέρεται από το **try** μπλοκ στο **catch**. Η μεταφορά αυτή γίνεται με τη λέξη **throw**.

Παράδειγμα

```
#include <iostream>
using namespace std;
int main() {
    char buffer[20]= " C++ language";
    int i, length;
    length=strlen(buffer);
    try
    {
        cout << " Write a number " ;
        cin >> i;
        if (i>length)
            throw i;
        if (i<0)
            throw "Negative value ...";
        cout << buffer[i] << endl;
    }
```

Παράδειγμα

```
catch (int n)
{
    cout << " Index error : " << n << endl;
}

catch (const char *mes)
{
    cout << mes << endl;
}
// Αυτή η εντολή θα εκτελεστεί σε κάθε περίπτωση
cout << " End of program ... " << endl;

}
```

Παράδειγμα

```
#include <iostream>
using namespace std;
int main() {
    int StudentAge;
    cout << "Student Age: ";
    cin >> StudentAge;
    try {
        if(StudentAge < 0)
            throw -1;
        cout << "\nStudent Age: " << StudentAge << "\n\n";
    }
    catch(...) {
        cout << "General Error...";
    }
    cout << "\n";
    return 0;
}
```

Throw

- ✓ Εάν εκτελέσετε αυτό το πρόγραμμα και πληκτρολογήσετε έναν θετικό ακέραιο για την ηλικία του μαθητή, το πρόγραμμα θα ανταποκριθεί εμφανίζοντας την ηλικία.
- ✓ Εάν εκτελέσετε το πρόγραμμα και πληκτρολογήσετε ένα γράμμα ή οποιοδήποτε χαρακτήρα, ο compiler θα εμφανίσει την ηλικία των φοιτητών ως 0.
- ✓ Όταν η εντολή **throw** εκτελεσθεί, είναι ένας τρόπος για να ζητηθεί από τον compiler να στείλει την εξαίρεση σε άλλο χειριστή. Στην πραγματικότητα, εάν δεν υπάρχει άλλος χειριστής, η επεξεργασία θα παραδοθεί στο λειτουργικό σύστημα. Σε αυτή την περίπτωση, το λειτουργικό σύστημα θα αναλάβει και θα εμφανίσει το δικό του μήνυμα πχ.«μη φυσιολογική λήξη του προγράμματος».

Παράδειγμα

```
#include <iostream>
using namespace std;
int main() {
    double Number1, Number2, Result;
    cout << "Please provide two numbers\n";
    try {
        cout << "First Number: ";
        cin >> Number1;
        cout << "Second Number: ";
        cin >> Number2;
        if( Number2 == 0 )
            throw -1;
        Result = Number1 / Number2;
        cout << "\n" << Number1 << " / " << Number2 << " = " <<
            Result << "\n\n" ;}
    catch(...){
    }
    return 0;}

```


Μηνύματα

- ✓ Κάθε φορά που εμφανίζεται `exception`, και κάθε φορά που χρησιμοποιείτε το `try` να δοκιμάσετε μια έκφραση, θα πρέπει να μεταφέρει τον έλεγχο σε ένα `catch`. Εκεί είναι όπου θα πρέπει να εμφανίζετε το δικό σας μήνυμα για το σφάλμα.

Παράδειγμα

```
#include <iostream>
int main(){
    int StudentAge;
    try {
        cout << "Student Age: ";
        cin >> StudentAge;
        if(StudentAge < 0)
            throw "Positive Number Required";
        cout << "\nStudent Age: " << StudentAge << "\n\n";
    }
    catch(const char* Message)
    {
        cout << "Error: " << Message;
    }

    cout << "\n";
    return 0;
}
```

Παράδειγμα

```
#include <iostream>
using namespace std;
int main()
{
    double Operand1, Operand2, Result;
    cout << "This program allows you to perform a division of two
                                                    numbers\n";
    cout << "To proceed, enter two numbers: ";
    try {
        cout << "First Number: ";
        cin >> Operand1;
        cout << "Second Number: ";
        cin >> Operand2;
```

Παράδειγμα/2

```
        if( Operand2 == 0 )
            throw "Division by zero not allowed";
        Result = Operand1 / Operand2;
        cout << "\n" << Operand1 << " / " << Operand2 << " = "
            << Result << "\n\n";
    }
    catch(const char* Str)
    {
        cout << "\nBad Operator: " << Str;
    }
    return 0;
}
```

catch

- ✓ Κατά την προετοιμασία για την αντιμετώπιση διαίρεσης με το μηδέν, η βασική ιδέα είναι να συγκρίνουμε τον παρονομαστή με 0.
- ✓ Η σύγκριση αυτή πρέπει να πραγματοποιηθεί με ένα `try`.
- ✓ Αν η σύγκριση είναι `true`, θα πρέπει να αποφεύχθει η λειτουργία με ένα **catch**.
- ✓ Το **catch** συνήθως χρησιμοποιείται για να εμφανίσει ένα μήνυμα.

Local Exception

- ✓ Το **catch** μπορεί να χρησιμοποιήσει οποιοδήποτε τύπο μεταβλητής, αρκεί να το ρυθμίσετε ανάλογα.
- ✓ Μπορείτε να στείλετε ένα ακέραιο και να εμφανίζει ένα σφάλμα, ανάλογα με τον ακέραιο που στάλθηκε.

Παράδειγμα

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    double Operand1, Operand2, Result;
```

```
    const char Operator = '/';
```

```
    cout << "This program allows you to perform a division of two  
                                         numbers\n";
```

```
    cout << "To proceed, enter two numbers\n";
```

```
    try {
```

```
        cout << "First Number: ";
```

```
        cin >> Operand1;
```

```
        cout << "Second Number: ";
```

```
        cin >> Operand2;
```

```
        // Find out if the denominator is 0
```

Παράδειγμα

```
if( Operand2 == 0 )
    throw 0;
Result = Operand1 / Operand2;
cout << "\n" << Operand1 << " / " << Operand2 << " = "
    << Result << "\n\n";
}
catch(const int n)
{
    cout << "\nBad Operator: Division by " << n << " not
        allowed\n\n";
}
return 0;
}
```


Πολλαπλά Exceptions

Τα παραδείγματα που έχουμε δει ως τώρα έχουν ασχοληθεί με μία μόνο εξαίρεση σε ένα πρόγραμμα. Τις περισσότερες φορές, ένα τυπικό πρόγραμμα θα έχει διαφορετικούς τύπους σφαλμάτων. Η γλώσσα C++ σας επιτρέπει να συμπεριλάβετε διάφορα **catch** :

```
try {  
    Code to Try  
}  
catch(Arg1) {  
    Μια Exception  
}  
catch(Arg2) {  
    Άλλη Exception  
}
```

Πολλαπλά Exceptions

- ✓ Κατά τον έλεγχο της ροής του προγράμματος, ο compiler εισέρχεται στο **try**.
- ✓ Αν δεν εμφανίζεται σφάλμα, το υπόλοιπο του try μπλοκ εκτελείται.
- ✓ Εάν εμφανίζεται μια εξαίρεση στο try μπλοκ, καθορίζετε το είδος του σφάλματος
- ✓ Ο μεταγλωττιστής βγαίνει από το try και εξετάζει το πρώτο **catch**
- ✓ Αν δεν ταιριάζει σφάλμα, ο compiler προχωρά στο επόμενο **catch**. Αυτό συνεχίζεται έως ότου ο compiler βρει **catch** που να ταιριάζει.

Παράδειγμα – Χωρίς έλεγχο

```
#include <iostream>

int main(){
    double Operand1, Operand2, Result;
    char Operator;
    cout << "This program allows you to perform an operation on
                                                    two numbers\n";
    cout << "To proceed, enter a number, an operator, and a
                                                    number:\n";

    cin >> Operand1 >> Operator >> Operand2;
    switch(Operator){
    case '+':
        Result = Operand1 + Operand2;
        break;
    case '-':
```

Παράδειγμα/2

```
        Result = Operand1 - Operand2;
        break;
    case '*':
        Result = Operand1 * Operand2;
        break;
    case '/':
        Result = Operand1 / Operand2;
        break;
    default:
        cout << "Bad Operation";
}
cout << "\n" << Operand1 << " " << Operator << " "
    << Operand2 << " = " << Result;
cout << "\n\n";
return 0;}
```

Παράδειγμα

```
#include <iostream>
#include <string>
using namespace std;
int main() {
```

```
    double Operand1, Operand2, Result;
    char Operator;
    cout << "This program allows you to perform an operation on
            two numbers\n";

    try {
        cout << "To proceed, enter a number, an operator, and
                a number:\n";
        cin >> Operand1 >> Operator >> Operand2;
        if(Operator != '+' && Operator != '-' &&
            Operator != '*' && Operator != '/')
            throw Operator;
        switch(Operator){
        case '+':
            Result = Operand1 + Operand2;
            break;
```

Έλεγχος του
τελεστή

```
        case '-':
            Result = Operand1 - Operand2;
            break;

        case '*':
            Result = Operand1 * Operand2;
            break;

        case '/':
            Result = Operand1 / Operand2;
            break;
    }
    cout << "\n" << Operand1 << " " << Operator << " "
         << Operand2 << " = " << Result;
}
catch(const char n) {
    cout << "\nOperation Error: " << n << " is not a valid
        operator";
}
cout << "\n\n";
return 0; }
```

Πολλαπλά Exceptions

- ✓ Φανταστείτε ότι ο χρήστης θέλει να εκτελέσει μια διαίρεση.
- ✓ Θα πρέπει να πείτε στο μεταγλωττιστή τι πρέπει να κάνετε αν ο χρήστης πληκτρολογήσει τον παρονομαστή ως 0 (ή 0.00).
- ✓ Αν συμβεί αυτό, η καλύτερη επιλογή είναι να εμφανιστεί ένα μήνυμα λάθους και να βγείτε.

Παράδειγμα

```
#include <iostream>
```

```
int main(){
```

```
    double Operand1, Operand2, Result;
```

```
    char Operator;
```

```
    cout << "This program allows you to perform a division of two  
            numbers\n";
```

```
    cout << "To proceed, enter two numbers\n";
```

```
    try {
```

```
        cout << "First Number: ";
```

```
        cin >> Operand1;
```

```
        cout << "Operator:   ";
```

```
        cin >> Operator;
```

Έλεγχος τελεστή και
διαίρεση με μηδέν

Παράδειγμα/2

```
cout << "Second Number: ";
cin >> Operand2;
if(Operator != '+' && Operator != '-' &&
    Operator != '*' && Operator != '/')
    throw Operator;
if(Operator == '/')
    if(Operand2 == 0)
        throw 0;
switch(Operator)
{
case '+':
```

Παράδειγμα/3

```
Result = Operand1 + Operand2;  
break;
```

```
case '-':
```

```
Result = Operand1 - Operand2;  
break;
```

```
case '*':
```

```
Result = Operand1 * Operand2;  
break;
```

```
case '/':
```

```
Result = Operand1 / Operand2;  
break;
```

Παράδειγμα/4

```
    }  
    cout << "\n" << Operand1 << " " << Operator << " "  
        << Operand2 << " = " << Result << "\n\n";  
}  
catch(const char n){  
    cout << "\nOperation Error: " << n << " is not a valid  
        operator\n\n";  
}  
catch(const int p){  
    cout << "\nBad Operation: Division by " << p << " not  
        allowed\n\n";  
}  
return 0;  
}
```

Παράδειγμα

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
int main() {
```

```
    char Number1[40], Number2[40];
```

```
    double Operand1, Operand2, Result;
```

```
    char Operator;
```

```
    cout << "This program allows you to perform a division of two  
            numbers\n";
```

```
    cout << "To proceed, enter two numbers\n";
```

```
    try {
```

```
        cout << "First Number: ";
```

```
        cin >> Number1;
```

```
        cout << "Operator:   ";
```

```
        cin >> Operator;
```

```
        cout << "Second Number: ";
```

Έλεγχος και των
τελεστών

Παράδειγμα/2

```
cin >> Number2;
for(int i = 0; i < strlen(Number1); i++)
    if( (!isdigit(Number1[i])) && (Number1[i] != '.') )
        throw Number1;
Operand1 = atof(Number1);
for(int j = 0; j < strlen(Number2); j++)
    if( (!isdigit(Number2[j])) && (Number2[j] != '.') )
        throw Number2;
Operand2 = atof(Number2);
if(Operator != '+' && Operator != '-' &&
    Operator != '*' && Operator != '/')
    throw Operator;
if(Operator == '/')
    if(Operand2 == 0)
        throw 0;
```

Παράδειγμα/3

```
switch(Operator)
{
case '+':
    Result = Operand1 + Operand2;
    break;
case '-':
    Result = Operand1 - Operand2;
    break;
case '*':
    Result = Operand1 * Operand2;
    break;
case '/':
    Result = Operand1 / Operand2;
    break;
}
```

Παράδειγμα/4

```
    cout << "\n" << Operand1 << " " << Operator << " "
        << Operand2 << " = " << Result << "\n\n";
}
catch(const int n)      {
    cout << "\nBad Operation: Division by " << n << " not
        allowed\n\n";
}
catch(const char n)     {
    cout << "\nOperation Error: " << n << " is not a valid
        operator\n\n";
}
catch(const char *BadOperand) {
    cout << "\nError: " << BadOperand << " is not a valid
        number\n\n";
}
```

Προσέξτε τη σειρά στα catch blocks

```
try
{
//statements
}
catch (...)
{
//statements
}
catch (int x)
{
//statements
}
```

Υπάρχει σφάλμα;

Εμφωλιασμένα Exceptions

- ✓ Η αριθμομηχανή, που έχουμε μελετήσει μέχρι στιγμής, προκειμένου να εκτελέσει οποιαδήποτε λειτουργία, ο compiler θα πρέπει πρώτα να βεβαιωθεί ότι ο χρήστης έχει εισάγει ένα έγκυρο τελεστή.
- ✓ Εφόσον ο τελεστής είναι ένα από αυτούς που περιμένουμε, ζητήσαμε επίσης τον compiler να ελέγξει την εγκυρότητα των αριθμών που εισήχθησαν.
- ✓ Ακόμη και αν τα δύο αυτά κριτήρια πληρούνται, ήταν δυνατόν ο χρήστης να εισαγάγει 0 για τον παρονομαστή.
- ✓ Το μπλοκ που χρησιμοποιείται για τον έλεγχο ενός μη μηδενικού παρονομαστή εξαρτάται από την εγκυρότητα των τελεστών.
- ✓ Η C++ επιτρέπει την εμφωλιάση εξαιρέσεων. Αυτό σημαίνει ότι μπορείτε να γράψετε μια εξαίρεση η οποία υπόκειται σε, άλλη.

Παράδειγμα

```
#include <iostream>
#include <string>
using namespace std;
int main(){
    char Number1[40], Number2[40];
    double Operand1, Operand2, Result;
    char Operator;
    cout << "This program allows you to perform an operation on
            two numbers\n";
    try {
        cout << "To proceed, enter\n";
        cout << "First Number: "; cin >> Number1;
        cout << "An Operator: "; cin >> Operator;
        cout << "Second Number: "; cin >> Number2;
        for(int i = 0; i < strlen(Number1); i++)
            if( (!isdigit(Number1[i])) && (Number1[i] != '.') )
                throw Number1;
    }
```

```
Operand1 = atof(Number1);  
for(int j = 0; j < strlen(Number2); j++)  
    if( (!isdigit(Number2[j])) && (Number2[j] != '.') )  
        throw Number2;
```

```
Operand2 = atof(Number2);  
if(Operator != '+' && Operator != '-' &&  
    Operator != '*' && Operator != '/')  
    throw Operator;
```

```
switch(Operator)  
{  
case '+':  
    Result = Operand1 + Operand2;  
    cout << "\n" << Operand1 << " + "  
        << Operand2 << " = " << Result;  
    break;  
case '-':
```

```
Result = Operand1 - Operand2;  
cout << "\n" << Operand1 << " - "  
      << Operand2 << " = " << Result;  
break;
```

```
case '*':
```

```
Result = Operand1 * Operand2;  
cout << "\n" << Operand1 << " * "  
      << Operand2 << " = " << Result;  
break;
```

```
case '/':
```

```
try {  
    if(Operand2 == 0)  
        throw "Division by 0 not  
              allowed";  
    Result = Operand1 / Operand2;  
    cout << "\n" << Operand1 << " / "
```

```
<< Operand2 << " = " <<  
Result;
```

```
}
```

```
catch(const char * Str){
```

```
cout << "\nBad Operation: " << Str;
```

```
}
```

```
break;
```

```
}
```

```
}
```

```
catch(const char n) {
```

```
cout << "\nOperation Error: " << n << " is not a valid  
operator";
```

```
}
```

```
catch(const char *BadOperand) {
```

```
cout << "\nError: " << BadOperand << " is not a valid  
number";
```

```
}
```

```
cout << "\n\n";
```

```
return 0;}
```

Exceptions και συναρτήσεις

- ✓ Μία από τις πιο αποτελεσματικές τεχνικές που χρησιμοποιούνται στον κώδικα είναι η απομόνωση αναθέσεων.
- ✓ Για παράδειγμα, η δήλωση switch μπορεί να γραφτεί όπως στο επόμενο παράδειγμα

Παράδειγμα

```
#include <iostream>
using namespace std;
double Calculator(const double N1, const double N2, const char p);
int main(){
    double Operand1, Operand2, Result;
    char Operator;
    cout << "This program allows you to perform a division of two
            numbers\n";
    cout << "To proceed, enter a number, an operator, and a
            number:\n";
    cin >> Operand1 >> Operator >> Operand2;
    Result = Calculator(Operand1, Operand2, Operator);
    cout << "\n" << Operand1 << " " << Operator << " "
            << Operand2 << " =" << Result;
    cout << "\n\n";
    return 0;}
```

```
double Calculator(const double Oper1, const double Oper2, const char
                Symbol){
```

Παράδειγμα/2

```
double Value;
switch(Symbol)
{
case '+':
    Value = Oper1 + Oper2;
    break;
case '-':
    Value = Oper1 - Oper2;
    break;
case '*':
    Value = Oper1 * Oper2;
    break;
case '/':
    Value = Oper1 / Oper2;
    break;
}
return Value;
}
```


Exceptions και συναρτήσεις

- ✓ Μπορείτε ακόμα να χρησιμοποιήσετε τις συνηθισμένες συναρτήσεις, μαζί με συναρτήσεις που χειρίζονται εξαιρέσεις

Παράδειγμα

```
#include <iostream>
#include <string>
using namespace std;
double Calculator(const double N1, const double N2, const char p);
int main() {
    char Number1[40], Number2[40];
    double Operand1, Operand2, Result;
    char Operator;
    cout << "This program allows you to perform an operation on
            two numbers\n";

    try {
        cout << "To proceed, enter\n";
        cout << "First Number: "; cin >> Number1;
        cout << "An Operator: "; cin >> Operator;
        cout << "Second Number: "; cin >> Number2;
        for(int i = 0; i < strlen(Number1); i++)
            if( (!isdigit(Number1[i])) && (Number1[i] != '.') )
```

```
        throw Number1;
Operand1 = atof(Number1);
for(int j = 0; j < strlen(Number2); j++)
    if( (!isdigit(Number2[j])) && (Number2[j] != '.') )
        throw Number2;
Operand2 = atof(Number2);
if(Operator != '+' && Operator != '-' &&
    Operator != '*' && Operator != '/')
    throw Operator;
if(Operator == '/')
    if(Operand2 == 0)
        throw 0;
Result = Calculator(Operand1, Operand2, Operator);
cout << "\n" << Operand1 << " " << Operator << " "
    << Operand2 << " = " << Result;
}
```

```
    catch(const int n){
        cout << "\nBad Operation: Division by " << n << " not
allowed";
    }
```

```
    catch(const char n){
        cout << "\nOperation Error: " << n << " is not a valid
operator";
    }
```

```
    catch(const char *BadOperand) {
        cout << "\nError: " << BadOperand << " is not a valid
number";
    }
```

```
    cout << "\n\n";
    return 0;
}
```

```
double Calculator(const double Oper1, const double Oper2, const char
                    Symbol) {
    double Value;
```

```
switch(Symbol) {  
  case '+':  
    Value = Oper1 + Oper2;  
    break;  
  case '-':  
    Value = Oper1 - Oper2;  
    break;  
  case '*':  
    Value = Oper1 * Oper2;  
    break;  
  case '/':  
    Value = Oper1 / Oper2;  
    break;  
}  
return Value;  
}
```

Exceptions και συναρτήσεις

- ✓ Όπως γίνεται στη `main()`, κάθε συνάρτηση ενός προγράμματος μπορεί να φροντίσει τις δικές της εξαιρέσεις.

Παράδειγμα

```
#include <iostream>
#include <string>
using namespace std;
void Calculator(const double N1, const double N2, const char p);
int main(){
    char Number1[40], Number2[40];
    double Operand1, Operand2, Result;
    char Operator;
    cout << "This program allows you to perform an operation on
            two numbers\n";
    try {
        cout << "To proceed, enter\n";
        cout << "First Number: "; cin >> Number1;
        cout << "An Operator: "; cin >> Operator;
        cout << "Second Number: "; cin >> Number2;
        for(int i = 0; i < strlen(Number1); i++)
            if( (!isdigit(Number1[i])) && (Number1[i] != '.') )
                throw Number1;
```

```
Operand1 = atof(Number1);
for(int j = 0; j < strlen(Number2); j++)
    if( (!isdigit(Number2[j])) && (Number2[j] != '.') )
        throw Number2;
```

```
Operand2 = atof(Number2);
if(Operator != '+' && Operator != '-' &&
    Operator != '*' && Operator != '/')
    throw Operator;
Calculator(Operand1, Operand2, Operator);
```

```
}
```

```
catch(const char n)    {
    cout << "\nOperation Error: " << n << " is not a valid
        operator";
```

```
}
```

```
catch(const char *BadOperand) {
    cout << "\nError: " << BadOperand << " is not a valid
        number";
```

```
}
```

```
cout << "\n\n";
```



```
return 0;
```

```
}
```

```
void Calculator(const double Oper1, const double Oper2, const char  
Symbol){
```

```
double Value;
```

```
switch(Symbol) {
```

```
case '+':
```

```
    Value = Oper1 + Oper2;
```

```
    cout << "\n" << Oper1 << " + "
```

```
        << Oper2 << " = " << Value;
```

```
    break;
```

```
case '-':
```

```
    Value = Oper1 - Oper2;
```

```
    cout << "\n" << Oper1 << " - "
```

```
        << Oper2 << " = " << Value;
```

```
    break;
```

```
case '*':
```

```
    Value = Oper1 * Oper2;
```

```
cout << "\n" << Oper1 << " * "  
      << Oper2 << " = " << Value;  
break;
```

```
case '/':
```

```
    try {  
        if(Oper2 == 0)  
            throw "Division by 0 not allowed";  
        Value = Oper1 / Oper2;  
        cout << "\n" << Oper1 << " / "  
              << Oper2 << " = " << Value;  
    }  
    catch(const char * Str) {  
        cout << "\nBad Operation: " << Str;  
    }  
    break;
```

```
}
```

```
}
```

Exceptions και συναρτήσεις

- ✓ Η απομόνωση αναθέσεων και η απόδοση του σε συναρτήσεις είναι ένα σημαντικό θέμα στον τομέα του προγραμματισμού.
- ✓ Σκεφτείτε ένα πρόγραμμα που χειρίζεται μια απλή εξαίρεση

Παράδειγμα

```
#include <iostream>
using namespace std;
int main(){
    double Operand1, Operand2, Result;
    char Operator = '/';
    cout << "This program allows you to perform a division of two
            numbers\n";
    try {
        cout << "To proceed, enter two numbers: ";
        cin >> Operand1 >> Operand2;
        if( Operand2 == 0 )
```

Παράδειγμα/2

```
        throw "Division by zero not allowed";
    Result = Operand1 / Operand2;
    cout << "\n" << Operand1 << " / " << Operand2 << " = "
        << Result;
}
catch(const char* Str)    {
    cout << "\nBad Operator: " << Str;
}
cout << "\n\n";
return 0;
}
```

Exceptions και συναρτήσεις

- ✓ Ένας από τους τρόπους που μπορείτε να χρησιμοποιήσετε συναρτήσεις σε εξαιρέσεις είναι να έχουμε μια κεντρική συνάρτηση που λαμβάνει τις μεταβλητές και τις στέλνει σε μια εξωτερική συνάρτηση.
- ✓ Η εξωτερική συνάρτηση ελέγχει την τιμή μιας μεταβλητής.
- ✓ Αν συμβεί εξαίρεση, η εξωτερική συνάρτηση στέλνει **throw**.
- ✓ Αυτό μπορεί να ληφθεί από τη συνάρτηση που έστειλε τη μεταβλητή.

Παράδειγμα

```
#include <iostream>
using namespace std;
void Division(const double a, const double b);
int main(){
    double Operand1, Operand2;
    cout << "This program allows you to perform a division of two
            numbers\n";
    try {
        cout << "To proceed, enter two numbers: ";
        cin >> Operand1 >> Operand2;
        Division(Operand1, Operand2);
    }
    catch(const char* Str)    {
```

Παράδειγμα/2

```
        cout << "\nBad Operator: " << Str;
    }
    cout << "\n\n";
    return 0;
}
void Division(const double a, const double b)
{
    double Result;
    if( b == 0 )
        throw "Division by zero not allowed";
    Result = a / b;
    cout << "\n" << a << " / " << b << " = " << Result;
}
```


Exceptions και συναρτήσεις

- ✓ Αν γράψετε μια συνάρτηση που φέρει μια εξαίρεση, μπορείτε να πληκτρολογήσετε το **throw** ακολουθούμενο από παρενθέσεις στη δεξιά πλευρά της συνάρτησης.

Παράδειγμα

```
#include <iostream>
using namespace std;
void Division(const double a, const double b) throw();
int main(){
    double Operand1, Operand2;
    cout << "This program allows you to perform a division of two
                                                    numbers\n";
    try {
        cout << "To proceed, enter two numbers: ";
        cin >> Operand1 >> Operand2;
        Division(Operand1, Operand2);
    }
    catch(const char* Str) {
```

Παράδειγμα/2

```
        cout << "\nBad Operator: " << Str;
    }
    cout << "\n\n";
    return 0;
}
void Division(const double a, const double b) throw() {
    double Result;
    if( b == 0 )
        throw;
    Result = a / b;
    cout << "\n" << a << " / " << b << " = " << Result;
}
```

Exceptions και συναρτήσεις

- ✓ Το **throw** που χρησιμοποιείτε έτσι πρέπει να έχει παρενθέσεις.
- ✓ Αν δεν παίρνει κανένα όρισμα, οι παρενθέσεις πρέπει να είναι κενές.
- ✓ Αν η συνάρτηση που καλείται από ένα μπλοκ **try** ρίξει συγκεκριμένη εξαίρεση, μπορείτε να το καθορίσετε αυτό στην παρένθεση του **throw**.

Παράδειγμα

```
#include <iostream>
#include <string>
using namespace std;
void Calculator(const double N1, const double N2,
               const char p) throw(const char*);
int main(){
    char Number1[40], Number2[40];
    double Operand1, Operand2;
    char Operator;
    cout << "This program allows you to perform an operation on
           two numbers \n";
    try {
```

Παράδειγμα/2

```
cout << "To proceed, enter \n";
cout << "First Number: "; cin >> Number1;
cout << "An Operator: "; cin >> Operator;
cout << "Second Number: "; cin >> Number2;
for(int i = 0; i < strlen(Number1); i++)
    if( (!isdigit(Number1[i])) && (Number1[i] != '.') )
        throw Number1;
Operand1 = atof(Number1);
for(int j = 0; j < strlen(Number2); j++)
```

Παράδειγμα/3

```
if( (!isdigit(Number2[j])) && (Number2[j] != '.') )  
    throw Number2;
```

```
Operand2 = atof(Number2);  
if(Operator != '+' && Operator != '-' &&  
    Operator != '*' && Operator != '/')  
    throw Operator;
```

```
try {  
    Calculator(Operand1, Operand2, Operator);
```

Παράδειγμα/4

```
    }  
    catch(const char * Str)  
    {  
        cout << "\nBad Operation: " << Str;  
    }  
}  
catch(const char n)  
{  
    cout << "\nOperation Error: " << n << " is not a valid  
        operator";  
}  
catch(const char *BadOperand)  
{
```


Παράδειγμα/5

```
        cout << "\nError: " << BadOperand << " is not a valid
                                                    number";
    }
    cout << "\n\n";
    return 0;
}
void Calculator(const double Oper1, const double Oper2,
                const char Symbol) throw(const char*)
{
    double Value;
```

Παράδειγμα/6

```
switch(Symbol)
{
case '+':
    Value = Oper1 + Oper2;
    cout << "\n" << Oper1 << " + "
        << Oper2 << " = " << Value;
    break;
case '-':
    Value = Oper1 - Oper2;
```

Παράδειγμα/7

```
cout << "\n" << Oper1 << " - "  
      << Oper2 << " = " << Value;  
break;
```

case '*':

```
Value = Oper1 * Oper2;
```

```
cout << "\n" << Oper1 << " * "  
      << Oper2 << " = " << Value;  
break;
```

Παράδειγμα/8

```
case '/':  
    if(Oper2 == 0)  
        throw "Division by 0 not allowed";  
  
    Value = Oper1 / Oper2;  
  
    cout << "\n" << Oper1 << " / "  
        << Oper2 << " = " << Value;  
    break;  
}  
}
```

Exceptions και συναρτήσεις

- ✓ Μια συνάρτηση μπορεί επίσης να κληθεί να εκτελέσει περισσότερες από μία δοκιμές και να ρίξει τελικά περισσότερο από μία εξαιρέσεις.
- ✓ Μια τέτοια συνάρτηση μπορεί (και πρέπει) να προγραμματιστεί για να ρίξει διάφορα είδη εξαιρέσεων.

Παράδειγμα

```
#include <iostream>
#include <string>
using namespace std;
void Calculator(const double N1, const double N2, const char p)
throw(const char*, const char);
double Validate(const char *N) throw(const char*);

int main() {
char Number1[40], Number2[40];
double Operand1, Operand2;
char Operator;

cout << "This program allows you to perform an operation on two
numbers\n";
```

Παράδειγμα/2

```
try {
    cout << "To proceed, enter \n";
    cout << "First Number: "; cin >> Number1;
    cout << "An Operator: "; cin >> Operator;
    cout << "Second Number: "; cin >> Number2;
    Operand1 = Validate(Number1);
    Operand2 = Validate(Number2);
    try {
        Calculator(Operand1, Operand2, Operator);
    }
    catch(const char * Str)
    {
        cout << "\nBad Operation: " << Str;
    }
}
```

Παράδειγμα/3

```
catch(const char n)
{
    cout << "\nOperation Error: " << n << " is not a valid operator";
}
catch(const char *BadOperand)
{
    cout << "\nError: " << BadOperand << " is not a valid number";
}
cout << "\n\n";
return 0;
}
```


Παράδειγμα/4

```
void Calculator(const double Oper1, const double Oper2, const char
Symbol) throw(const char*, const char)
{
double Value;
if(Symbol != '+' && Symbol != '-' && Symbol != '*' && Symbol != '/')
throw Symbol;
switch(Symbol)
{
case '+':
    Value = Oper1 + Oper2;
    cout << "\n" << Oper1 << " + " << Oper2 << " = " << Value;
    break;
```

Παράδειγμα/5

case '-':

```
Value = Oper1 - Oper2;
```

```
cout << "\n" << Oper1 << " - " << Oper2 << " = " << Value;
```

```
break;
```

case '*':

```
Value = Oper1 * Oper2;
```

```
cout << "\n" << Oper1 << " * " << Oper2 << " = " << Value;
```

```
break;
```

case '/':

```
if(Oper2 == 0) throw "Division by 0 not allowed";
```

```
Value = Oper1 / Oper2;
```

```
cout << "\n" << Oper1 << " / " << Oper2 << " = " << Value;
```

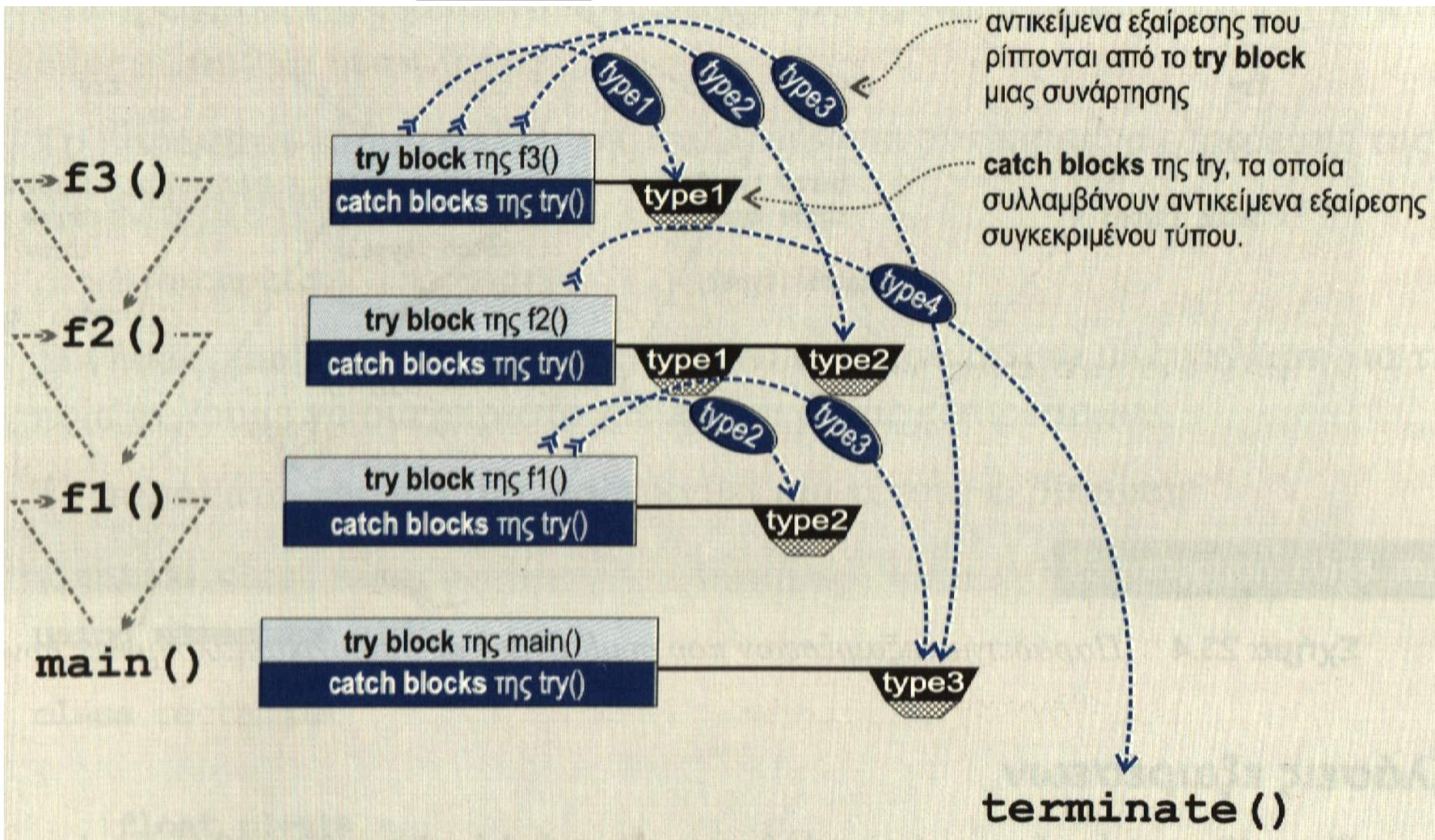
```
break;
```

```
}}
```

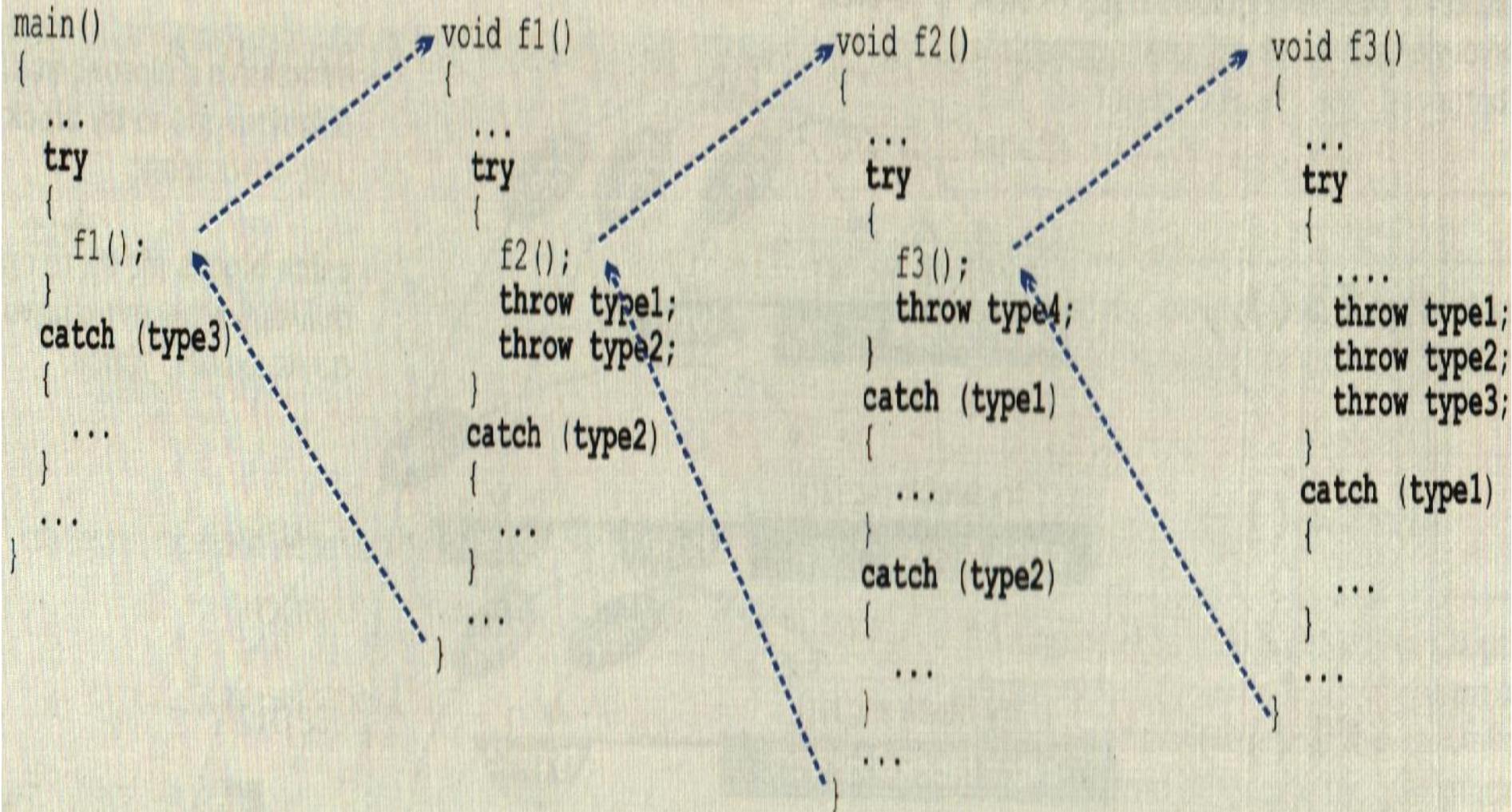
Παράδειγμα/6

```
double Validate(const char* N) throw(const char*)
{
    double Valid;
    for(int i = 0; i < strlen(N); i++)
        if( (!isdigit(N[i])) && (N[i] != '.') )
            throw N;
    Valid = atof(N);
    return Valid;
}
```

Εξαιρέσεις σε διαδοχικά καλούμενες συναρτήσεις



Εξαιρέσεις σε διαδοχικά καλούμενες συναρτήσεις



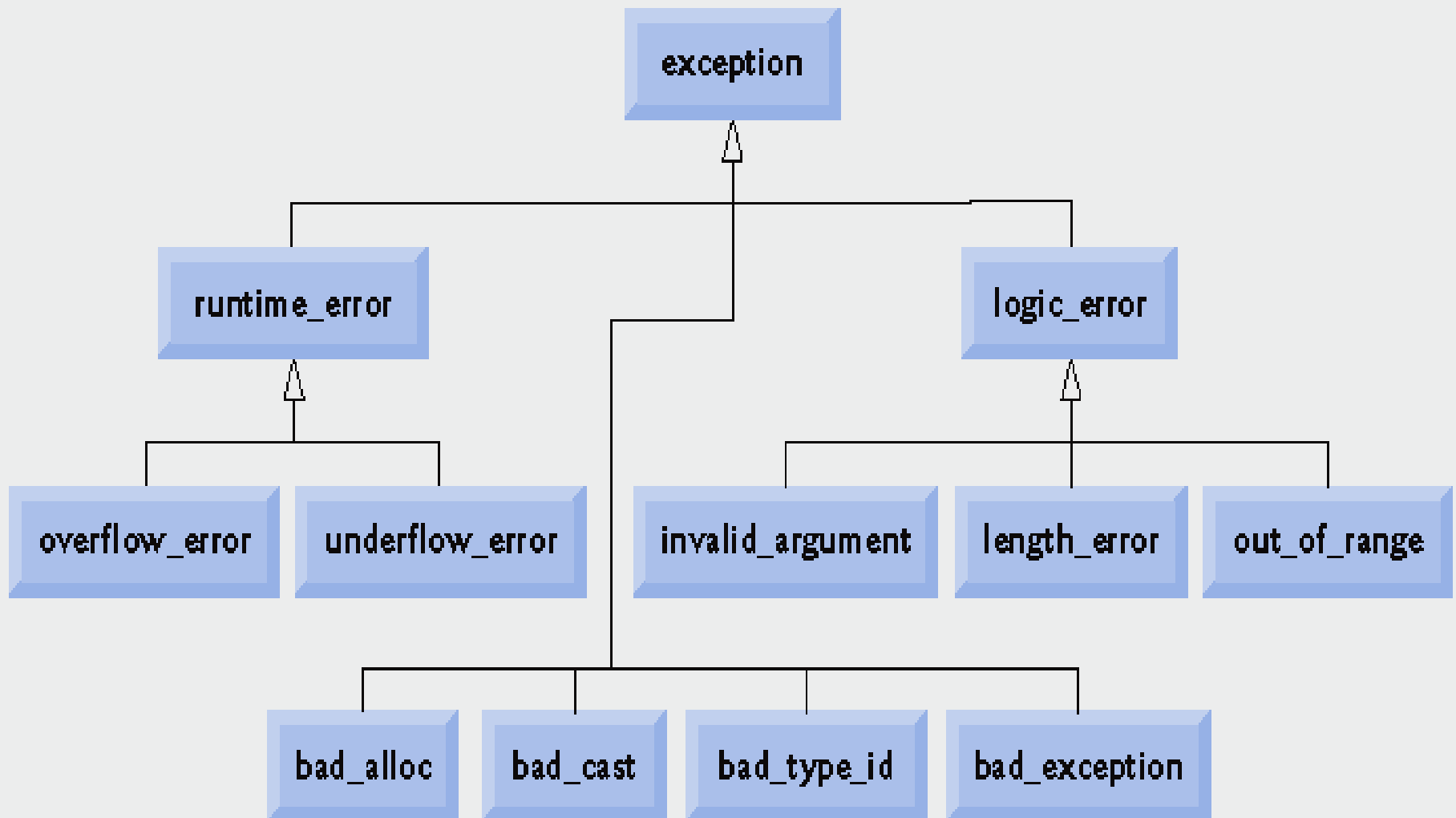
Κλάσεις εξαιρέσεων

- ✓ Γνωρίζουμε ότι ένα τμήμα σύλληψης (catch-block) συλλαμβάνει μόνο αντικείμενα εξαίρεσης συγκεκριμένου τύπου. Αν λοιπόν περιοριστούμε στους βασικούς τύπους δεδομένων, η ποικιλία των εξαιρέσεων που μπορούμε να δημιουργήσουμε αλλά και να συλλάβουμε, είναι πολύ περιορισμένη.
- ✓ Μπορούμε να δημιουργήσουμε προσαρμοσμένες κλάσεις τις οποίες να χρησιμοποιούμε μόνο για τον χειρισμό των εξαιρέσεων (κλάσεις εξαιρέσεων)
- ✓ Μπορούμε να χρησιμοποιήσουμε τις ενσωματωμένες κλάσεις εξαιρέσεων της C++

Ενσωματωμένες Κλάσεις εξαιρέσεων

- ✓ Η C++ διαθέτει αρκετές ενσωματωμένες κλάσεις χειρισμού εξαιρέσεων οι οποίες μπορούν να χρησιμοποιηθούν στα προγράμματα μας.
- ✓ Όλες οι κλάσεις εξαιρέσεων παράγονται άμεσα ή έμμεσα από τη βασική κλάση `exception`.
- ✓ Μια από τις πιο συχνά χρησιμοποιούμενες τέτοιες κλάσεις είναι η κλάση `bad_alloc`

Η κλάση exception



Παράδειγμα

```
int *ptr, num;
bool done=false;
while (!done) {
cout  << " Δώσε πλήθος θέσεων ";
cin >> num;
try
{
ptr=new int[num];
done=true;
}
}
```

Παράδειγμα

```
catch (bad_alloc)
{
    cout << "Δεν υπάρχει τόση διαθέσιμη μνήμη" << endl;
    cout << "Δώστε νέο μέγεθος" << endl;
}

}

...
```

Δημιουργία δικών μας εξαιρέσεων

Οι κλάσεις εξαιρέσεων μπορούν να μην περιέχουν κανένα μέλος:

```
class my_exception  
{  
  
  
};
```

Η δημιουργία μιας εξαίρεσης, με χρήση της παραπάνω κλάσης, γίνεται με την εντολή:

```
throw my_exception(); ← Δημιουργεί ένα αντικείμενο  
εξαίρεσης της κλάσης exception
```

Παράδειγμα

class tooBig ← Κλάση εξαίρεσης για έλεγχο μεγέθους
σχήματος

{

public:

float emv;

tooBig(float e) {emv=e;}

};

Δημιουργία εξαίρεσης:

...

if (emvado()>1000) throw tooBig(emvado());

Παράδειγμα

Τμήμα σύλληψης:

```
catch(tooBig obj)
{
cout << "Too big rectangle, with area : " << obj.emv << endl;
}
```

Μπορούμε να επεκτείνουμε την exception

```
#include <exception>
class FileException : public exception {
public:
virtual const char* what() const throw() {
    return "error accessing stream";
}
}
```

```
char safeReadByte(istream &source) {
    if (!source)
        throw FileException();
    return source.get();
}
```