C++: Από τη Θεωρία στην Εφαρμογή

Εξαιρέσεις

Έλεγχος Σφαλμάτων

- Όταν εκτελείται ένα πρόγραμμα είναι πιθανό να εμφανιστούν σφάλματα που, αν το πρόγραμμα δεν τα χειριστεί με κατάλληλο τρόπο, μπορεί να προκαλέσουν τον βίαιο τερματισμό του
- Η παραδοσιακή τεχνική διαχείρισης σφαλμάτων είναι να υπάρχει διάσπαρτος κώδικας με το κάθε τμήμα του προγράμματος να χειρίζεται τα δικά του σφάλματα
- Για παράδειγμα, όταν χρησιμοποιούμε μία συνάρτηση πρέπει να ελέγχουμε αν εκτελέστηκε με επιτυχία. Αν αποτύχει, ο προγραμματιστής πρέπει να έχει προσθέσει κώδικα που να συλλαμβάνει το λάθος και να το χειρίζεται
- Αν η συνάρτηση επιστρέφει τιμή, ένας συνηθισμένος τρόπος για τον έλεγχο λαθών είναι:

Παράδειγμα

```
k = f();
if(k == ERROR_VALUE_1)
{
    ...
}
else if(k == ERROR_VALUE_2)
{
    ...
}
else if(k == ERROR_VALUE_N)
{
    ...
}
else // Δεν συνέβη λάθος.
{
    ...
}
```

Σημειώστε ότι αν η συνάρτηση δεν επιστρέφει τιμή, μπορούμε να χρησιμοποιήσουμε ένα όρισμα δείκτη ή αναφορά για να επιστραφεί το αποτέλεσμα της εκτέλεσης της συνάρτησης

Εξαιρέσεις

- Η C++ παρέχει τον μηχανισμό των **εξαιρέσεων** (exceptions) ως μία εναλλακτική τεχνική αντιμετώπισης προβλημάτων
- Οι εξαιρέσεις αναφέρουν λάθη που συμβαίνουν κατά την **εκτέλεση** του προγράμματος
- Ειδικότερα, με τις εξαιρέσεις επιτρέπεται ο **διαχωρισμός** της **ανίχνευσης** των σφαλμάτων από τον **χειρισμό** τους
- Δηλαδή, το λάθος συμβαίνει σε ένα σημείο του προγράμματος και η **διαχείρισή** σε κάποιο άλλο σημείο
- Η **φιλοσοφία** του μηχανισμού είναι να υπάρχουν συγκεκριμένα σημεία στο πρόγραμμα για την διαχείριση των σφαλμάτων και όχι διάσπαρτος κώδικας που να τα διαχειρίζεται
- Έτσι, ο κώδικας γίνεται πιο απλός και η αντιμετώπιση προβλημάτων διαβάζεται και ελέγχεται πιο εύκολα
- Ας δούμε την σύνταξη για ένα συνηθισμένο παράδειγμα για τη δημιουργία και σύλληψη εξαιρέσεων:

Παράδειγμα

```
f()
{
    try
    {
            g();
    }
    try(type exc_1)
    {
            ...
    }
    catch(type exc_2)
    {
            ...
    }
    catch(...) // Συλλαμβάνει κάθε τύπο εξαίρεσης.
    {
            ...
    }
}

g()
{
    ...
    if(error_occurs)
            throw exception; // Η g() δημιουργεί μία εξαίρεση.
    ...
}
```

Εξαιρέσεις

- Η ιδέα είναι ότι αν συμβεί κάποιο λάθος σε μία συνάρτηση (π.χ. `g()`) και η συνάρτηση δεν μπορεί να το διαχειριστεί, μπορεί να χρησιμοποιήσει την εντολή `throw` για να **δημιουργήσει** μία εξαίρεση που να ενημερώνει την συνάρτηση που την κάλεσε για το λάθος
- Για να μπορεί να **συλλάβει** *(catch)* την εξαίρεση, η κλήση της συνάρτησης που μπορεί να δημιουργήσει εξαιρέσεις πρέπει να τοποθετηθεί σε ένα τμήμα που αρχίζει με την εντολή `try`
- Ο κώδικας που χειρίζεται τις εξαιρέσεις τοποθετείται μετά την εντολή `try` σε ένα ή περισσότερα τμήματα σύλληψης που αρχίζουν με την εντολή `catch`
- Σημειώστε ότι δεν επιτρέπεται να εισάγεται κώδικας μεταξύ του `try` τμήματος και του πρώτου `catch` τμήματος ή μεταξύ διαδοχικών `catch` τμημάτων, το πρόγραμμα δεν θα μεταγλωττιστεί
- Κάθε `catch` τμήμα καθορίζει τον τύπο της εξαίρεσης που μπορεί να συλλάβει. Ο τύπος δηλώνεται μετά την `catch` λέξη μέσα στις παρενθέσεις και μπορεί να είναι ένας βασικός τύπος αλλά και ένας τύπος που έχει οριστεί από τον προγραμματιστή

Παρατηρήσεις

- Η **βασική** έννοια των εξαιρέσεων είναι όταν συμβαίνει ένα «εξαιρετικό» γεγονός σε ένα τμήμα του προγράμματος (π.χ. στη `g()`), αυτό το τμήμα να μπορεί να **πληροφορήσει** ένα άλλο (π.χ. την `f()`) για αυτό το «**εξαιρετικό**» γεγονός και να του μεταφέρει πληροφορία σχετικά με αυτό
- Ο προγραμματιστής πρέπει να **καθορίσει** τι είναι αυτό το «εξαιρετικό». Για παράδειγμα, μία απόπειρα διαίρεσης με το μηδέν είναι κάτι «εξαιρετικό»
- Σημειώστε ότι το «εξαιρετικό» δεν είναι απαραίτητο να είναι πάντα καταστροφικό, ο προγραμματιστής θα πρέπει να καθορίσει την σοβαρότητά του, μπορεί να είναι απλά η αδυναμία εκτέλεσης κάποιας εργασίας
- Ουσιαστικά, η δημιουργία εξαίρεσης αποτελεί έναν τρόπο ώστε το τμήμα του προγράμματος το οποίο δεν μπορεί να διαχειριστεί τοπικά το «εξαιρετικό» γεγονός (π.χ. `g()`) να το **προωθήσει** σε ένα ανώτερο επίπεδο (π.χ. `f()`), το οποίο ίσως να μπορεί να το διαχειριστεί

Σύλληψη Εξαιρέσεων

- Η εντολή `throw` δημιουργεί μία εξαίρεση και το όρισμά της καθορίζει τον τύπο της εξαίρεσης
- Η `throw` τερματίζει την εκτέλεση της τρέχουσας συνάρτησης (π.χ. `g()`) και η εκτέλεση του προγράμματος μεταβαίνει στην καλούσα συνάρτηση (π.χ. `f()`)
- Εκεί, το πρόγραμμα ελέγχει διαδοχικά τις `catch` εντολές για να δει αν υπάρχει κάποια που να ταιριάζει με τον τύπο της εξαίρεσης. Για παράδειγμα, η εντολή `throw 20;` δημιουργεί μία εξαίρεση τύπου `int` και μεταβιβάζει την τιμή `20`
- Αν υπάρχει μία `catch` εντολή για εξαιρέσεις ακέραιου τύπου, π.χ. `catch(int a)`, θα συλλάβει τη συγκεκριμένη εξαίρεση, η τιμή του `a` θα γίνει `20` και θα εκτελεστεί ο κώδικας του τμήματος
- Σημειώστε ότι αν γράψουμε `catch(double a)`, η εξαίρεση δεν θα συλληφθεί γιατί ο μεταγλωττιστής δεν πραγματοποιεί τις συνηθισμένες αριθμητικές μετατροπές. Οι μετατροπές που υποστηρίζονται είναι περιορισμένες

Σύλληψη Εξαιρέσεων

- Το `catch` τμήμα μπορεί να περιέχει κώδικα που να διορθώνει το πρόβλημα, αν αυτό μπορεί να διορθωθεί
- Αν όχι, ο προγραμματιστής μπορεί να αποφασίσει να προσθέσει κώδικα που να τερματίζει το πρόγραμμα. Τα υπόλοιπα τμήματα δεν ελέγχονται
- Σημειώστε ότι δεν είναι υποχρεωτικό να δηλώσουμε ονόματα παραμέτρων, π.χ. `catch(int)`
- Συνήθως, το `throw` όρισμα που μεταβιβάζεται στο `catch` τμήμα παρέχει πληροφορία που σχετίζεται με την εξαίρεση, οπότε η `catch` παράμετρος χρησιμεύει για την αποθήκευση αυτής της πληροφορίας

Σύλληψη Εξαιρέσεων

- Το τμήμα σύλληψης `catch(...)` είναι **προαιρετικό** και **συλλαμβάνει** όλους τους τύπους των εξαιρέσεων
- Αν υπάρχει και η εξαίρεση δεν συλληφθεί από κάποιο προηγούμενο `catch` τμήμα θα συλληφθεί από αυτό. Μοιάζει κάπως με την `default` περίπτωση στην εντολή `switch`
- Σημειώστε ότι θα πρέπει να τοποθετηθεί στο **τέλος** των `catch` τμημάτων. Αν τοποθετηθεί σε άλλη θέση, επειδή συλλαμβάνει όλες τις εξαιρέσεις, οι `catch` εντολές που μπορεί να υπάρχουν μετά από αυτό δεν θα ελεγχθούν
- Αν λείπει, και η εξαίρεση δεν συλληφθεί από **κανένα** τμήμα, τότε η εξαίρεση **προωθείται** προς την `main()`, μέχρι να βρεθεί κάποιο τμήμα σύλληψης που να ταιριάζει με τον τύπο της. Αν δεν βρεθεί κάποιο τέτοιο τμήμα, το πρόγραμμα **τερματίζει**
- Αν δεν δημιουργηθεί κάποια εξαίρεση στο `try` τμήμα, οι `catch` εντολές **παρακάμπτονται** και η εκτέλεση του προγράμματος συνεχίζει με την εντολή μετά το τελευταίο `catch` τμήμα
- Ας δούμε ένα παράδειγμα. Το παρακάτω πρόγραμμα διαβάζει ένα συνθηματικό και ελέγχει αν είναι έγκυρο

Παράδειγμα

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

int check_pswd(const string& str);

int main()
{
	string pswd;
	while(1)
	{
		try
		{
			cout << "Enter password: ";
			cin >> pswd;
			if(pswd == "end")
				break;
			check_pswd(pswd);
		}
		catch(const char *msg)
		{
			cout << msg;
			continue; /* Αν έλειπε, θα
	εκτελούνταν η εντολή μετά το τελευταίο catch τμήμα. */
		}
		catch(int code)
		{
			if(code == 10)
				cout << "Error:
	Password must contain three digits at least\n";
			else if(code == 20)
				cout << "Error:
	Password must contain one special character at least\n";
			continue;
		}
		catch(...)
		{
		       cout << "Error: Another exception\n";
			continue;
		}
	       cout << "Password " << pswd << " is accepted\n";
	}
	return 0;
}
```

```
int check_pswd(const string& str)
{
	bool found;
	int i, len, dig;

	len = str.size();
	if(len < 6)
	   throw "Error: Short length\n";

	dig = 0;
	found = false;
	for(i = 0; i < len; i++)
	{
	if(str[i] >= '0' && str[i] <= '9')
			dig++;
	if(str[i] == '!' || str[i] == '$' ||
str[i] == '#')
			found = true;
	}
	if(dig < 3)
			throw 10;
	if(found == false)
			throw 20;
	return 0;
}
```

Παράδειγμα

- Αρχικά, η `check_pswd()` ελέγχει το μήκος του συνθηματικού. Αν είναι μικρότερο από έξι χαρακτήρες η εντολή `throw` δημιουργεί μία εξαίρεση και μεταβιβάζει ένα κυριολεκτικό αλφαριθμητικό στην καλούσα συνάρτηση, δηλαδή, στη `main()`
- Άρα, ο τύπος της εξαίρεσης είναι `const char*`. Η `throw` τερματίζει την `check_pswd()` και η εκτέλεση του προγράμματος μεταβαίνει στη `main()`
- Γενικά, όταν δημιουργείται μία εξαίρεση, η στοίβα με τις κλήσεις των συναρτήσεων ξετυλίγεται ώστε να μεταφερθεί ο έλεγχος του προγράμματος στη συνάρτηση που μπορεί να διαχειριστεί αυτόν τον τύπο της εξαίρεσης
- Αφού η κλήση της `check_pswd()` έγινε μέσα από `try` τμήμα, το πρόγραμμα μπορεί να συλλάβει τις εξαιρέσεις που δημιουργεί
- Αν είχε γίνει έξω από `try` τμήμα, οι εξαιρέσεις δεν θα μπορούσαν να συλληφθούν

Παράδειγμα

- Τώρα, το πρόγραμμα ελέγχει αν υπάρχει κάποια `catch` εντολή που να ταιριάζει με τον τύπο της εξαίρεσης. Έτσι, το τμήμα με τύπο `const char*` συλλαμβάνει την εξαίρεση και εκτελείται ο κώδικάς του
- Το αλφαριθμητικό "Error: Short length\n" μεταβιβάζεται στην παράμετρο `msg` και αυτό εμφανίζεται στην οθόνη
- Με την εντολή `continue` το πρόγραμμα συνεχίζει με την επόμενη επανάληψη του βρόχου και ο χρήστης εισάγει νέο συνθηματικό. Θα μπορούσαμε στη θέση της `continue` να έχουμε `break` ή `return` για τον τερματισμό του προγράμματος ή κάποια άλλη προσέγγιση, όπως ότι αν ο χρήστης εισάγει τρεις φορές ένα μη έγκυρο συνθηματικό το πρόγραμμα να τερματίζει
- Σε κάθε τμήμα σύλληψης, ο προγραμματιστής **αποφασίζει** για τον τρόπο που θα χειριστεί την εξαίρεση
- Σημειώστε ότι αν δεν υπήρχε το `const char*` τμήμα, η εξαίρεση θα συλλαμβανόταν από το `catch(...)` τμήμα και θα εκτελούνταν ο κώδικάς του. Αν δεν υπήρχε κανένα από τα δύο, η εκτέλεση του προγράμματος θα τερματιζόταν

Παράδειγμα

- Αν το μήκος του συνθηματικού είναι αποδεκτό, η `check_pswd()` ελέγχει αν περιέχει λιγότερα από τρία ψηφία
- Αν ναι, δημιουργεί μία εξαίρεση τύπου `int`. Τότε, η εκτέλεση της `check_pswd()` τερματίζεται και το `int` τμήμα στη `main()` συλλαμβάνει την εξαίρεση. Η τιμή του `code` γίνεται ίση με `10` και το πρόγραμμα εμφανίζει το αντίστοιχο μήνυμα
- Αν ο αριθμός των ψηφίων είναι έγκυρος, η `check_pswd()` ελέγχει αν το συνθηματικό περιέχει κάποιον από τους χαρακτήρες `!`, `$` ή `#`
- Αν όχι, δημιουργεί μία εξαίρεση τύπου `int`, όπου στο τμήμα σύλληψης το `code` γίνεται ίσο με `20` και το πρόγραμμα εμφανίζει το αντίστοιχο μήνυμα
- Όπως και πριν, η εντολή `continue` προκαλεί την επόμενη επανάληψη του βρόχου. Σημειώστε ότι, σε αντίθεση με την `return` η οποία μπορεί να επιστρέψει έναν τύπο δεδομένων, η `throw` μπορεί να επιστρέψει διαφορετικούς τύπους

Παράδειγμα

- Αν το συνθηματικό είναι αποδεκτό, η `check_pswd()` δεν δημιουργεί κάποια εξαίρεση και επιστρέφει την τιμή `0`
- Ο λόγος που επιλέχθηκε η συνάρτηση να επιστρέφει τιμή είναι για να δείτε ότι μία συνάρτηση μπορεί να δημιουργεί εξαιρέσεις αλλά και να επιστρέφει τιμή, η οποία και αυτή να δηλώνει αν εκτελέστηκε σωστά ή όχι
- Στη συνέχεια, το πρόγραμμα παρακάμπτει τα `catch` τμήματα και εμφανίζει το αποδεκτό συνθηματικό
- Το πρόγραμμα εκτελείται μέχρι ο χρήστης να εισάγει το `end`. Τότε, η εντολή `break` θα τερματίσει την εκτέλεση του βρόχου
- Τέλος, με το συγκεκριμένο παράδειγμα βλέπουμε ότι οι `break` και `continue` εντολές μέσα σε ένα `try-catch` τμήμα λειτουργούν με τον συνηθισμένο τρόπο

Εξαιρέσεις με Τύπο Κλάση

- Εκτός από τη δημιουργία εξαιρέσεων με βασικούς τύπους δεδομένων (π.χ. `int`), μπορεί να δημιουργηθούν εξαιρέσεις που ο τύπος τους να είναι **κλάση**
- Γενικά, αυτή είναι η συνηθέστερη επιλογή
- Ένα πλεονέκτημα αυτής της επιλογής είναι ότι μπορούμε να χρησιμοποιήσουμε διαφορετικά αντικείμενα για να ξεχωρίζουν τα διαφορετικά είδη προβλημάτων που δημιουργούν εξαιρέσεις
- Ένα άλλο πλεονέκτημα είναι ότι ένα αντικείμενο μπορεί να περιέχει όση πληροφορία χρειάζεται για να εντοπιστούν οι αιτίες που προκάλεσαν την εξαίρεση
- Η πρότυπη βιβλιοθήκη παρέχει αρκετές κλάσεις εξαιρέσεων τις οποίες μπορούμε να χρησιμοποιήσουμε στα προγράμματά μας Βρίσκονται στον χώρο ονομάτων `std` και όλες προέρχονται από την κλάση `exception`
- Για παράδειγμα, ας αλλάξουμε το προηγούμενο πρόγραμμα ώστε η εξαίρεση να μεταβιβάζει ένα αντικείμενο:

Παράδειγμα

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

class Err_Rpt
{
private:
      int code;
      string msg;
public:
      Err_Rpt(): code(0), msg("") {}
      Err_Rpt(int c, const char *m) : code(c), msg(m) {}
      void show() const {cout << "C:" << code << ' ' << msg;}
};
int check_pswd(const string& str);

int main()
{
      string pswd;
      while(1)
      {
                  try
                  {
                              cout << "Enter password: ";
                              cin >> pswd;
                              check_pswd(pswd);
                              break;
                  }
                  catch(const Err_Rpt& err)
                  {
                              err.show();
                  }
                  catch(int code)
                  {
                  cout << "C:" << code << ' ' << "Error:
      Password must contain one special character at least\n";
                  }
                  catch(...)
                  {
                        cout << "Error: Another exception\n";
                  }
      }
      cout << "Password " << pswd << " is accepted\n";
      return 0;
}
```

```
int check_pswd(const string& str)
{
            bool found;
            int i, len, dig;

            len = str.size();
            if(len < 6)
                  throw Err_Rpt(5, "Error: Short
length\n"); /* Δημιουργία εξαίρεσης με τύπο
αντικείμενο της κλάσης Err_Rpt. */

            dig = 0;
            found = false;
            for(i = 0; i < len; i++)
            {
            if(str[i] >= '0' && str[i] <= '9')
                        dig++;
            if(str[i] == '!' || str[i] == '$' ||
str[i] == '#')
                        found = true;
            }
            if(dig < 3)
                        throw Err_Rpt(10, "Error:
Password must contain three digits at
least\n");
            if(found == false)
                        throw 20;
            return 0;
}
```

Παράδειγμα

- Αν ο χρήστης εισάγει ένα έγκυρο συνθηματικό, η εντολή `break` τερματίζει τον βρόχο και το πρόγραμμα το εμφανίζει
- Η `check_pswd()` δημιουργεί μία εξαίρεση με τύπο `int` και άλλες δύο με τύπο **αντικείμενο** της κλάσης `Err_Rpt`
- Κάθε αντικείμενο αρχικοποιείται με τιμές που υποδεικνύουν τον τύπο του λάθους
- Αυτές οι εξαιρέσεις συλλαμβάνονται από το αντίστοιχο `catch` τμήμα, το οποίο καλεί την `show()` του αντικειμένου που μεταβιβάζεται για να εμφανίσει την πληροφορία που περιέχεται στα πεδία του
- Σημειώστε ότι στη θέση της `int` εξαίρεσης θα μπορούσαμε να δημιουργήσουμε και άλλη μία εξαίρεση με `Err_Rpt` τύπο. Για παράδειγμα, μπορούμε να γράψουμε `throw Err_Rpt();` όπου το αντικείμενο που μεταβιβάζεται δεν περιέχει πληροφορία
- Απλά, αυτό το πρόγραμμα αποτελεί ένα παράδειγμα που **συνδυάζει** τη δημιουργία και σύλληψη βασικών τύπων και τύπων που ορίζονται από τον χρήστη

Εξαιρέσεις και Κληρονομικότητα

- Για παράδειγμα, στο παρακάτω πρόγραμμα έχουμε τρεις κλάσεις εξαιρέσεων που σχετίζονται μεταξύ τους με μία σχέση κληρονομικότητας:

```
#include <iostream>

class Err1
{
public:
	virtual void show() const {std::cout << "Err1\n";}
};

class Err2 : public Err1
{
public:
	virtual void show() const {std::cout << "Err2\n";}
};

class Err3 : public Err2
{
public:
	virtual void show() const {std::cout << "Err3\n";}
};

void f()
{
	throw Err3();
}

int main()
{
	try
	{
		f();
	}
	catch(const Err1& err)
	{
		err.show();
	}
	return 0;
}
```

Εξαιρέσεις και Κληρονομικότητα

- Επειδή η παράμετρος στο `catch` τμήμα είναι αναφορά στη βασική κλάση, ταιριάζει με οποιαδήποτε εξαίρεση παράγωγης κλάσης
- Επειδή η `show()` είναι εικονική, η επιλογή της `show()` που θα εκτελεστεί γίνεται με βάση τον τύπο του αντικειμένου στον οποίο αναφέρεται η `err`. Έτσι, το πρόγραμμα εμφανίζει `Err3`
- Αν δεν χρησιμοποιούσαμε αναφορά θα καλούνταν πάντα η `show()` της `Err1` και το πρόγραμμα θα εμφάνιζε `Err1`, ανεξάρτητα από το αντικείμενο-όρισμα της `throw`
- Όταν δημιουργούνται εξαιρέσεις, η καλύτερη πρακτική είναι η μεταβίβαση αντικειμένου. Ο γενικός κανόνας είναι να μεταβιβάζεται το αντικείμενο κατ'αξία και να συλλαμβάνεται μέσω αναφοράς

Εξαιρέσεις και Κληρονομικότητα

- Αν θέλουμε να χειριστούμε με διαφορετικό τρόπο την κάθε εξαίρεση, προσθέτουμε ξεχωριστές `catch` εντολές για τον κάθε τύπο. **Προσοχή** όμως στη σειρά. Η σειρά των `catch` τμημάτων θα πρέπει να είναι η **αντίστροφη** της παραγωγής των κλάσεων, δηλαδή, από την τελευταία παράγωγη προς την βασική. Για παράδειγμα:

```
int main()
{
        int i;

        try
        {
                f();
        }
        catch(const Err3& err)
        {
                i = 3;
        }
        catch(const Err2& err)
        {
                i = 2;
        }
        catch(const Err1& err)
        {
                i = 1;
        }
        return 0;
}
```

- Αν το `Err1&` τμήμα σύλληψης ήταν πρώτο, θα συλλάμβανε όλες τις `Err1`, `Err2`, και `Err3` εξαιρέσεις και η τιμή του `i` θα ήταν πάντα `1`
- Σε μία ιεραρχική σχέση κληρονομικότητας με κλάσεις εξαιρέσεων, το πρώτο τμήμα σύλληψης **πρέπει** να αναφέρεται στην τελευταία παράγωγη κλάση και το τελευταίο τμήμα στη βασική κλάση

Προώθηση Εξαιρέσεων

- Το σύστημα διαχείρισης των εξαιρέσεων μπορεί να είναι πολυεπίπεδο
- Δηλαδή, το κάθε επίπεδο να διαχειρίζεται όσα προβλήματα μπορεί και τα υπόλοιπα να τα αφήνει για τα ανώτερα επίπεδα
- Έτσι, αν μία εξαίρεση δεν συλληφθεί από την τρέχουσα συνάρτηση, η εξαίρεση **προωθείται** στη συνάρτηση που την κάλεσε, και από εκεί σε αυτήν που την κάλεσε, σε όλη τη διαδρομή μέχρι την `main()`, μέχρι να βρεθεί κάποιο τμήμα σύλληψης που να συλλάβει την εξαίρεση
- Αν η `main()` δεν συλλάβει την εξαίρεση, το πρόγραμμα, εξ'ορισμού θα τερματιστεί
- Αυτή η διαδικασία ονομάζεται ξεδίπλωμα στοίβας (stack unwinding). Για παράδειγμα, ας δούμε τον παρακάτω κώδικα:

```
int main()
{
 ...
 try
 {
    f();
 }
 catch(int)
 {
    ...
 }
}
```

```
f()
{
 ...
 try
 {
        g();
 }
 catch(double)
 {
        ...
 }
}
```

```
g()
{
 ...
 try
 {
        h();
 }
 catch(const char*)
 {
        ...
 }
}
```

```
h()
{
   ...
throw 20;
}
```

Προώθηση Εξαιρέσεων

- Η εξαίρεση στην `h()` προωθείται σε υψηλότερα επίπεδα και τελικά συλλαμβάνεται από το τμήμα σύλληψης της `main()`, το οποίο μπορεί να χειριστεί εξαιρέσεις ακεραίου τύπου. Αν αντί για `throw 20;` γράψουμε `throw 1.2;` αυτή η εξαίρεση θα συλληφθεί από το τμήμα σύλληψης της `f()`, ενώ αν γράψουμε `throw "Test";` θα συλληφθεί από το τμήμα σύλληψης της `g()`. Αν έχει δηλωθεί κάποια κλάση `A` και `a` είναι αντικείμενό της και γράψουμε `throw a;` αυτή η εξαίρεση θα προωθηθεί μέχρι την `main()` και επειδή δεν υπάρχει `catch` τμήμα για τον τύπο `A` το πρόγραμμα θα τερματιστεί
- Αυτό το παράδειγμα δείχνει επίσης μία πολύ σημαντική ιδιότητα της `throw`. Ενώ η `return` μεταφέρει την εκτέλεση του προγράμματος στην πρώτη εντολή μετά την κλήση της συνάρτησης, η `throw` μεταφέρει την εκτέλεση σε όλη τη διαδρομή μέχρι να συναντήσει το πρώτο `catch` τμήμα που μπορεί να συλλάβει την εξαίρεση

Επανακατάθεση Εξαίρεσης

- Όταν ένα `catch` τμήμα συλλαμβάνει μία εξαίρεση είναι πιθανό να μη μπορεί να την διαχειριστεί ή να αποφασίσει ότι είναι καλύτερα να την διαχειριστεί ένα ανώτερο επίπεδο
- Σε αυτή την περίπτωση μπορούμε να **επανακαταθέσουμε** την εξαίρεση, ώστε να προωθηθεί σε **ανώτερο** επίπεδο, πιο σχετικό με τον τύπο της εξαίρεσης, και να αναλάβει αυτό την διαχείρισή της
- Για να κάνουμε κάτι τέτοιο γράφουμε **μόνο** το όνομα `throw`, χωρίς κάποιο όρισμα
- Η `throw` πρέπει να βρίσκεται μέσα σε ένα `catch` τμήμα ή μέσα σε μία συνάρτηση που να καλείται (άμεσα ή έμμεσα σε μία αλυσίδα κλήσεων) από ένα `catch` τμήμα. Αλλιώς, η κλήση της `throw` προκαλεί τον **τερματισμό** του προγράμματος
- Η εξαίρεση που επανακατατίθεται είναι η **αρχική** εξαίρεση που είχε συλληφθεί. Για παράδειγμα, στο επόμενο πρόγραμμα, το αρχικό όρισμα της `throw` μεταβιβάζεται σε όλη τη διαδρομή μέχρι το τμήμα σύλληψης που τελικά θα διαχειριστεί την εξαίρεση:

Παράδειγμα

```
#include <iostream>

void f();
void g();

int main()
{
	try
	{
		f();
	}
	catch(int a)
	{
		std::cout << "int exception is caught: " << a << '\n';
	}
	return 0;
}
void f()
{
	try
	{
		g();
	}
	catch(int) /* Αφού το τμήμα σύλληψης δεν χρησιμοποιεί το όρισμα δεν δηλώνω αντίστοιχη
	μεταβλητή. */
	{
		std::cout << "int exception is rethrown\n";
		throw; // Επανακατάθεση της εξαίρεσης.
	}
	std::cout << "f() terminates\n"; /* Αφού η throw επανακαταθέτει την εξαίρεση αυτή η εντολή
	δεν θα εκτελεστεί. */
}

void g()
{
	 throw 10;
}
```

Παράδειγμα

- Η `int` εξαίρεση που δημιουργεί η `g()` συλλαμβάνεται από το `catch` τμήμα της `f()`. Η `f()` την επανακαταθέτει και αυτή συλλαμβάνεται στη `main()`. Στη `main()` μεταβιβάζεται και το αρχικό όρισμα της `throw`. Άρα, το πρόγραμμα εμφανίζει:

```
int exception is rethrown
int exception is caught: 10
```

- Αυτό το πρόγραμμα αποτελεί ένα παράδειγμα ένθετων `try` τμημάτων. Όπως είδαμε και στο προηγούμενο παράδειγμα, ένα `try` τμήμα, μαζί με `catch` τμήματά του, μπορεί να τοποθετηθεί μέσα σε ένα άλλο `try` τμήμα
- Σε αυτό το παράδειγμα, το εξωτερικό τμήμα είναι αυτό στο `main()`, ενώ το εσωτερικό τμήμα είναι αυτό στην `f()`
- Κάθε `try` τμήμα συσχετίζεται με το δικό του σύνολο `catch` τμημάτων, το οποίο χειρίζεται τις εξαιρέσεις που ενδέχεται να δημιουργηθούν μέσα σε αυτό
- Αν κανένα δεν ταιριάζει με τον τύπο της εξαίρεσης, θα την χειριστούν τα `catch` τμήματα του εξωτερικού `try` τμήματος
- Για παράδειγμα, αν στην `f()` αλλάξουμε τον τύπο στο `catch` τμήμα από `int` σε `double`, τότε η εξαίρεση θα προωθηθεί στο εξωτερικό `try` τμήμα στη `main()`. Εκεί, το `catch` τμήμα για τον τύπο `int` θα συλλάβει την εξαίρεση