Χειρισμός Εξαιρέσεων

Εξαιρέσεις

- Εξαίρεση
  - Δείχνουν ότι κάποιο πρόβλημα προέκυψε στην εκτέλεση του προγράμματος
  - Κάτι μη φυσιολογικό
- Χειρισμός Εξαιρέσεων
  - Resolve exceptions
  - Το πρόγραμμα μπορεί να συνεχίσει την λειτουργία
    - Controlled termination
  - Για δημιουργία fault-tolerant programs

Try – Catch Blocks

- Κώδικας C++

```
try {
  κώδικα που μπορεί να προκαλέσει εξαίρεση
}
catch (exceptionType){
  κώδικας για χειρισμό εξαιρέσεων
}
```

  - Το **`try`** μπλοκ περιέχει κώδικα ο οποίος μπορεί να προκαλέσει εξαίρεση
  - Τα **`catch`** μπλοκ (1 ή περισσότερα):
    - Λαμβάνουν και χειρίζονται τις εξαιρέσεις
    - Μέσω παραμέτρου μπορούν να προσπελάσουν το αντικείμενο εξαίρεσης.
    - Αποσιωπητικά **`catch (…)`** : σύλληψη οποιουδήποτε τύπου εξαίρεσης
      - Χρησιμοποιείται συνήθως ως το τελευταίο catch block

Άμεση έγερση εξαιρέσεων

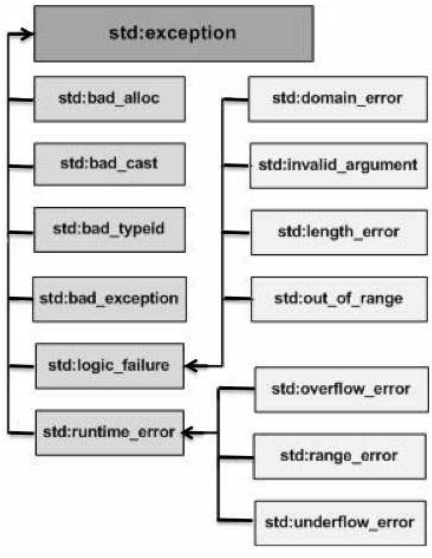
- Εντολή **`throw`**
  - Εγείρει μια εξαίρεση
    - Χρήση όταν προκύπτει το σφάλμα
  - Μπορούμε με «πετάξουμε» με την throw σχεδόν οτιδήποτε (αντικείμενα-εξαιρέσεις, built-in τύπους όπως integer, κτλ.)
    - **`throw myObject;`**
    - **`throw 5;`**
- Αντικείμενα-Εξαιρέσεις
  - Έχουν ως κλάση βάσης την **`exception`** (**`<exception>`**)
  - Ο δημιουργός μπορεί να έχει ένα αλφαριθμητικό για περιγραφή του σφάλματος.
  - Η περιγραφή μπορεί να ανακτηθεί μέσω της *virtual* μεθόδου **`what()`**

Παράδειγμα, throw int

```
1  // exceptions
2  #include <iostream>
3  using namespace std;
4
5  int main () {
6    try
7    {
8      throw 20;
9    }
10   catch (int e)
11   {
12     cout << "An exception occurred. Exception Nr. " << e << '\n';
13   }
14   return 0;
15 }
```

```
An exception occurred. Exception Nr. 20
```

C++ Standard Exceptions



#include <stdexcept>

Exception	Description
std::exception	An exception and parent class of all the standard C++ exceptions.
std::bad_alloc	This can be thrown by new.
std::bad_cast	This can be thrown by dynamic_cast.
std::bad_exception	This is useful device to handle unexpected exceptions in a C++ program
std::bad_typeid	This can be thrown by typeid.
std::logic_error	An exception that theoretically can be detected by reading the code.
std::domain_error	This is an exception thrown when a mathematically invalid domain is used
std::invalid_argument	This is thrown due to invalid arguments.
std::length_error	This is thrown when a too big std::string is created
std::out_of_range	This can be thrown by the at method from for example a std::vector and std::bitset<>::operator[]().
std::runtime_error	An exception that theoretically can not be detected by reading the code.
std::overflow_error	This is thrown if a mathematical overflow occurs.
std::range_error	This is occured when you try to store a value which is out of range.
std::underflow_error	This is thrown if a mathematical underflow occurs.

Ροή εκτέλεσης

- Σημείο έγερσης εξαίρεσης (Throw point)
  - Το σημείο στο μπλοκ **`try`** όπου εγείρεται εξαίρεση
  - Αν χειριστεί η εξαίρεση
    - Το πρόγραμμα προσπερνάει τον υπόλοιπο κώδικα του **`try`** μπλοκ
    - Συνεχίζεται η λειτουργία μετά τα **`catch`** μπλοκ
  - αν δεν χειριστεί η εξαίρεση
    - Τερματίζεται η λειτουργία
      - `std::terminate`→ `std::abort`
- Αν δεν προκύψει εξαίρεση
  - Το πρόγραμμα αγνοεί τα **`catch`** blocks.

Custom exceptions

```
#include <iostream>
#include <stdexcept>
using namespace std;
class DivideByZeroException : public out_of_range {
  public:
    DivideByZeroException(int n, string m)
              : numerator(n), out_of_range(m){}
    int numerator;
};

double divide( int numerator, int denominator ) {
  if ( denominator == 0 )
    throw DivideByZeroException(numerator,"attempted to divide with zero");
  return static_cast< double >( numerator ) / denominator;
}

int main(){
  int a,b;
  try{
    while ( cin >> a >> b )
      divide(a,b);
  }
  catch (out_of_range e){
    cout << e.what();
  }
}
```

Αν και δεν είναι υποχρεωτικό, εδώ φτιάχνουμε δικό μας τύπο εξαίρεσης, κληρονομώντας από τον υπάρχοντα τύπο out_of_range.
Μέσω του δημιουργού μπορούμε να ορίσουμε την πληροφορία που θα αποθηκεύουμε στην εξαίρεση μέσω των ορισμάτων του. (πχ εδώ μπορούμε να κρατάμε τον αριθμητή της πράξης).
Στον δημιουργό της κλάσης out_of_range μπορούμε να περνάμε μήνυμα σχετικό με το σφάλμα το οποία θα είναι προσπελάσιμο μέσω της μεθόδου what() που έχει η κλάση exception και παράγωγες της.

Με κλήση του δημιουργού, δημιουργούμε εξαίρεση του τύπου που φτιάξαμε και την πετάμε με την εντολή throw.

Στο catch κομμάτι χειριζόμαστε εξαιρέσεις τύπου out_of_range ή παράγωγες της (όπως αυτή που φτιάξαμε) . **Πέρασμα με αναφορά;**

```
catch (Exception &e){
    cout << e.what();
}
```

Λίστα εξαιρέσεων συνάρτησης

- Ορίζουμε την λίστα των εξαιρέσεων που μπορεί να εγερθούν στην συνάρτηση αυτή
  - Also called throw list

```
int someFunction( double value )
  throw ( ExceptionA, ExceptionB, ExceptionC )
{
   // σώμα συνάρτησης
}
```

  - Η συνάρτηση μπορεί να πετάξει εξαιρέσεις τύπου **`ExceptionA`**, **`ExceptionB`**, and **`ExceptionC`** (ή παράγωγες αυτών)
    - Αν προκύψει άλλου είδους εξαίρεση (και δεν χειριστεί μέσα στο σώμα της συνάρτησης με κάποιο catch block) προκύπτει απροσδόκητο σφάλμα και τερματίζει το πρόγραμμα (`std::unexpected` → `std::terminate`)
  - Αν δεν ορίσουμε λίστα throw, μπορεί να πετάξει οποιαδήποτε
  - Αν ορίσουμε κενή λίστα throw, δεν μπορεί να πετάξει καμία εξαίρεση
- **C++ 11:** κατάργηση throw list!
  - Χρήση **`noexcept`** αντί για throw()

Επανέγερση

- Επανέγερση εξαίρεσης (rethrowing)
  - Χρησιμοποιείται σε catch μπλοκ, όταν δεν μπορεί να χειριστεί η εξαίρεση ώστε να επανεγερθεί
  - Can rethrow exception to another handler
    - Goes to next enclosing **`try`** block
    - Corresponding **`catch`** blocks try to handle
- Για να ξαναπετάξουμε εξαίρεση μέσα σε catch block καλούμε την εντολή **`throw`** (χωρίς όρισμα)

Παραδείγματα

```
void function() throw(int){
  throw 5;
}

int main(){
  try{ function(); }
  catch(int){
     cout << " handled";
  }
}
```

handled
(εγείρεται εξαίρεση αποδεκτού τύπου στην function).
Η main έχει κατάλληλο catch μπλοκ χειρισμού του τύπου οπότε χειρίζεται και τερματίζει κανονικά η λειτουργία

```
void function() throw(int){
  throw 'k';
}

int main(){
  try{ function(); }
  catch(int){
     cout << " handled";
  }
}
```

Τερματισμός με σφάλμα
Μη αποδεκτός τύπος εξαίρεσης.
Δεν έχει οριστεί ο τύπος char στην λίστα throw της συνάρτησης.

```
void function() throw(int){
  try{
    throw 'k';
  }
  catch(...){
    cout << "handled internally"; }
}
int main(){
  try{ function(); }
  catch(int){
     cout << " handled";
  }
}
```

handled internally
(Εγείρεται εξαίρεση μη αποδεκτού τύπου, ωστόσο χειρίζεται εσωτερικά).
Η main δεν «ενημερώνεται» ότι προέκυψε εξαίρεση στην function

Παραδείγματα

```
void function() throw(int, char){
  try{
    throw 'k';
  }
  catch(...){
    cout << "handled internally";
    throw;
  }
}

int main(){
  try{ function(); }
  catch(int){
     cout << " handled";
  }
}
```

handled internally
Τερματίζει με σφάλμα
(Η εξαίρεση συλλαμβάνεται αρχικά εσωτερικά, ωστόσο με την throw την πετάει ξανά στο παραπάνω επίπεδο (main) , το οποίο δεν έχει κατάλληλο catch χειρισμού της.

```
void function() throw(int, char){
  try{
    throw 'k';
  }
  catch(...){
    cout << "handled internally";
    throw;
  }
}

int main(){
  try{ function(); }
  catch(char ch){
     cout <<endl<< ch << " handled";
  }
}
```

handled internally
k handled
Η εξαίρεση χειρίζεται και εσωτερικά και στην main

Τεχνικές Χειρισμού Εξαιρέσεων

- Αγνόηση εξαιρέσεων
  - Τυπικό για προσωπικό (όχι εμπορικό) λογισμικό
  - Το πρόγραμμα μπορεί να αποτυγχάνει
- Τερματισμός Προγράμματος
  - Συνήθως είναι κατάλληλο
  - Δεν είναι κατάλληλο για κρίσιμες εφαρμογές
- Set error indicators
  - Unfortunately, may not test for these when necessary
- Test for error condition
  - Call exit (**<cstdlib>**) and pass error code

Διαφορές Exceptions C++/Java

1. Στην C++ κάθε είδους αντικείμενο μπορεί να εγερθεί ως εξαίρεση. Στη Java μόνο αντικείμενα της κλάσης Throwable (υπ. Exception)

```
#include <iostream>
using namespace std;
int main()
{
   int x = -1;

   // some other stuff
   try {
      // some other stuff
      if( x < 0 )
      {
         throw x;
      }
   }
   catch (int x ) {
      cout << "Exception occurred: thrown value is " << x << endl;
   }
   getchar();
   return 0;
}
```

Διαφορές Exceptions C++/Java

2. Στη C++ υπάρχει το catch-all catch(…). Στην Java catch (Exception e)

```
#include <iostream>
using namespace std;
int main()
{
   int x = -1;
   char *ptr;

   ptr = new char[256];

   // some other stuff
   try {
      // some other stuff
      if( x < 0 )
      {
         throw x;
      }
      if(ptr == NULL)
      {
         throw " ptr is NULL ";
      }
   }
   catch (...) // catch all
   {
      cout << "Exception occurred: exiting "<< endl;
      exit(0);
   }

   getchar();
   return 0;
}
```

Διαφορές Exceptions C++/Java

3. Στη C++ δεν υπάρχει block *finally* όπως στη Java. Η αποδέσμευση πόρων (πρέπει να) γίνεται στον destructor (ιδίωμα **RAII**)
4. Στη C++ όλες οι exceptions είναι unchecked. Ο compiler δεν θα διαμαρτυρηθεί αν δεν γίνει χειρισμός μιας exception, όπως στην Java.
5. Στη Java υπάρχει ξεχωριστή λέξη για τη λίστα εξαιρέσεων, η throws. Η C++ χρησιμοποιεί την ίδια λέξη throw(). Μπορεί επίσης να εγείρει μια εξαίρεση χωρίς καθόλου throw.