Εξαιρέσεις

#18

Τμήμα Πληροφορικής και Τηλεπικοινωνιών (Άρτα)

Πανεπιστήμιο Ιωαννίνων

Γκόγκος Χρήστος

Χειρισμός εξαιρέσεων στη C++

- Εξαίρεση (exception): ένα λάθος ή μια προβληματική κατάσταση
  - π.χ. διαίρεση με το μηδέν, πρόσβαση σε NULL δείκτη, …
- Χειρισμός εξαιρέσεων (exception handling): αφορά την αντιμετώπιση του λάθους ή της προβληματικής κατάστασης.
  - Η C++ έχει ενσωματωμένους μηχανισμούς για χειρισμό εξαιρέσεων.
    - Αν δεν χρησιμοποιηθεί ο ενσωματωμένος στη γλώσσα μηχανισμός χειρισμού εξαιρέσεων, τα λάθη και οι προβληματικές καταστάσεις μπορούν να αντιμετωπιστούν με προσθήκη ελέγχων (ifs) μέσα στον κώδικα.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample1.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample2.cpp

Γιατί να χρησιμοποιήσει κανείς χειρισμό εξαιρέσεων;

- Ο έλεγχος λαθών με χρήση εντολών `if` αναμειγνύει το χειρισμό λαθών με τον κώδικα που επιτελεί το κύριο έργο του προγράμματος.
- Πολλά από τα πιθανά λάθη συμβαίνουν πολύ σπάνια.
  - Ο κώδικας χειρισμού αυτών των σπάνιων περιπτώσεων δεν θα πρέπει να αναμειγνύεται με την κύρια λογική του προγράμματος καθώς η κατανόηση και η αποσφαλμάτωση του κώδικα καθίστανται δύσκολες.
- Με το χειρισμό εξαιρέσεων ο κώδικας που χειρίζεται τις εξαιρέσεις τοποθετείται ξεχωριστά από την κύρια λογική του προγράμματος και έτσι βελτιώνεται η αναγνωσιμότητα του κώδικα.
- Ο χειρισμός εξαιρέσεων, συχνά βελτιώνει και την ανοχή σε σφάλματα (fault tolerance) του προγράμματος καθώς αποτελεί ένα συστηματικότερο τρόπο για το χειρισμό λαθών.

Πότε πρέπει να χρησιμοποιείται ο χειρισμός εξαιρέσεων;

- Ο χειρισμός εξαιρέσεων δεν αποτελεί πάντα τον πλέον κατάλληλο τρόπο για το χειρισμό λαθών.
  - Π.χ. ο παραδοσιακός τρόπος (με χρήση ifs) είναι καλύτερος για έλεγχο εισόδου.
- Πότε ο χειρισμός εξαιρέσεων αποτελεί καλύτερη λύση;
  - Χειρισμός προβλημάτων που συμβαίνουν σπάνια.
  - Χειρισμός προβλημάτων που δεν αφορούν ένα μόνο μπλοκ κώδικα.
  - Όταν πρέπει να χρησιμοποιηθεί μια ομοιόμορφη τεχνική χειρισμού λαθών από πολλούς προγραμματιστές που ο καθένας εργάζεται στο δικό του τμήμα κώδικα.

Ο χειρισμός εξαιρέσεων στη C++

- Τα μπλοκς `try-throw-catch`

```
try
{
… κώδικας που πρέπει να ελεγχθεί για εξαιρέσεις
… πιθανές προκλήσεις εξαιρέσεων (throw exceptions)
}
catch (type1 catch_parameter_1)
{… κώδικας χειρισμού εξαίρεσης τύπου type1}
catch (type2 catch_parameter_2)
{… κώδικας χειρισμού εξαίρεσης τύπου type2}
```

Το μπλοκ try

- Συντακτικό:

```
try
{
… η κύρια λογική, πιθανά προκαλεί εξαιρέσεις (throw exceptions)
}
```

- Περιέχει τον κώδικα που πρόκειται να εκτελεστεί όταν η εκτέλεση θα πραγματοποιηθεί χωρίς προβλήματα
  - Ωστόσο, ο κώδικας μπορεί να προκαλέσει εξαιρέσεις, άρα θα πρέπει να πραγματοποιηθεί «δοκιμαστική» λειτουργία (`try`).
  - Αν κάτι μη συνηθισμένο συμβεί, αυτό υποδηλώνεται με την πρόκληση μιας εξαίρεσης (throw exception).

Η εντολή throw

- Σύνταξη εντολής throw:
  - `throw exception_for_value_to_be_thrown;`
- Σημασία:
  - Υποδηλώνει ότι συνέβη μια εξαίρεση.
  - Αν μια εντολή throw εκτελεστεί, το μπλοκ `try` αυτόματα τερματίζει.
  - Το πρόγραμμα προσπαθεί να ταιριάξει την εξαίρεση με κάποιο από τα `catch` μπλοκς που ακολουθούν το `try` μπλοκ (που περιέχουν τον κώδικα χειρισμού της κάθε εξαίρεσης).
  - Το ταίριασμα γίνεται με βάση τον τύπο της τιμής που γίνεται `throw`.
  - Αν βρεθεί κάποιο ταίριασμα, εκτελείται ο κώδικας στο `catch` μπλοκ του.
  - Μόνο ένα `catch` μπλοκ μπορεί να ταιριάξει το πολύ.
  - Το πρόγραμμα συνεχίζει την εκτέλεσή του με τον κώδικα μετά το τελευταίο `catch` μπλοκ.
- Τόσο η τιμή της εξαίρεσης όσο και η ροή ελέγχου μεταβιβάζονται (γίνονται `throw`) στο `catch` μπλοκ που αποτελεί το χειριστή της εξαίρεσης.

Το catch μπλοκς

- Ένα ή περισσότερα `catch` μπλοκς ακολουθούν το `try` μπλοκ.
- Κάθε `catch` μπλοκ έχει έναν τύπο παραμέτρου.
- Κάθε `catch` μπλοκ είναι ένας χειριστής εξαιρέσεων (που χειρίζεται εξαιρέσεις ενός τύπου).
- `catch (type catch_block_parameter)`
  `{ … κώδικας χειρισμού εξαίρεσης }`
- Η `catch_block_parameter` συλλαμβάνει (catches) την τιμή της εξαίρεσης που προκαλείται και η οποία μπορεί να χρησιμοποιηθεί στον κώδικα χειρισμού της εξαίρεσης.
- Η εξαίρεση που προκαλείται ταιριάζει με μια από τις παραμέτρους των `catch` μπλοκς.
  - Αν αυτό δεν συμβεί τότε έχουμε την κατάσταση στην οποία δεν γίνεται `catch` μια εξαίρεση (un-caught exception).

Σύνοψη του `try-throw-catch`

- Σε κανονικές συνθήκες (δηλαδή στο συνηθέστερο σενάριο) εκτελείται το `try` μπλοκ και στη συνέχεια ο κώδικας μετά το τελευταίο catch μπλοκ.
- Αν το `try` μπλοκ προκαλέσει μια εξαίρεση (δηλαδή εκτελεστεί μια εντολή `throw`) τότε:
  - Το `try` μπλοκ σταματά αμέσως μετά την εντολή throw.
  - Ο κώδικας στο `catch` μπλοκ ξεκινά να εκτελείται με τη throw τιμή να περνά ως παράμετρος του κατάλληλου `catch` μπλοκ.
  - Όταν ολοκληρώσει την εκτέλεση του το catch μπλοκ, εκτελείται ο κώδικας μετά το `catch` μπλοκ.
- Αν προκληθεί μια εξαίρεση (γίνει `throw`) αλλά δεν ταιριάξει με κάποιο από τα διαθέσιμα catch μπλοκς τότε η συνάρτηση τερματίζει.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample3.cpp

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample4.cpp

Πολλαπλά `throws` και `catches`

- Κάθε `catch` μπλοκ χειρίζεται έναν τύπο εξαίρεσης.
- Μπορεί να χρειαστούμε πολλά catch μπλοκς.
- Όταν η τιμή που γίνεται `throw` δεν είναι σημαντική για το χειρισμό της εξαίρεσης, τότε το όνομα της παραμέτρου στο `catch` μπλοκ μπορεί να παραληφθεί συμπληρώνοντας στη θέση του τρεις τελείες.
  - Π.χ. `catch(int) {…}`
- Το `catch(…)` συλλαμβάνει οποιαδήποτε εξαίρεση, και μπορεί να χρησιμοποιηθεί ως ο προκαθορισμός χειριστής εξαιρέσεων.
- Το `catch(…)` τοποθετείται ως τελευταίο `catch` μπλοκ.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample5.cpp

Κλάσεις εξαιρέσεων

- Συχνά ορίζονται κλάσεις με σκοπό το χειρισμό εξαιρέσεων.
- Μπορούν να χρησιμοποιηθούν έτσι ώστε για κάθε κλάση εξαίρεσης να υπάρχει διαφορετική κλάση.
- Οι κλάσεις εξαιρέσεων είναι κανονικές κλάσεις που απλά χρησιμοποιούνται με σκοπό το χειρισμό εξαιρέσεων.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample6.cpp

Οι ενσωματωμένες κλάσεις εξαιρέσεων του προτύπου της C++

- Στην C++ υπάρχει η standard βιβλιοθήκη που περιλαμβάνει ορισμένες προκαθορισμένες κλάσεις εξαιρέσεων.
- Η βασική κλάση είναι η exception και βρίσκεται στο αρχείο επικεφαλίδας exception:
  - ```
    #include <exception>
    using std::exception;
    ```
- Ο προγραμματιστής μπορεί να ορίσει δικές του κλάσεις εξαιρέσεων που να κληρονομούν από την κλάση `std::exception`.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample7.cpp
    ```

Παράγωγες κλάσεις εξαιρέσεων της std::exception

- Ορισμένες παράγωγες κλάσεις της `std::exception` είναι:
  - `bad_alloc` ➔ προκαλείται από το new όταν δημιουργείται πρόβλημα κατά τη δέσμευση μνήμης.
  - `bad_cast` ➔ προκαλείται σε περίπτωση αποτυχίας του `dynamic_cast`.
  - `logic_error` ➔ σφάλμα σχετιζόμενο με την εσωτερική λογική του κώδικα.
  - `out_of_range` ➔ σφάλμα εκτός περιοχής.
  - `invalid_argument` ➔ μη έγκυρο όρισμα.
  - `runtime_error` ➔ σφάλμα κατά το χρόνο εκτέλεσης.
  - `domain_error` ➔ σφάλμα που υποδηλώνει ότι χρησιμοποιήθηκε μια τιμή εκτός του εύρους αποδεκτών τιμών.
- `#include <stdexcept>`

Πρόκληση μιας εξαίρεσης μέσα από μια συνάρτηση

- Μέχρι τώρα, οι εξαιρέσεις προκαλούνται (γίνονται `throw`) και συλλαμβάνονται (γίνονται `catch`) στο ίδιο επίπεδο κώδικα.
- Στην πράξη, τα προγράμματα χωρίζονται σε τμήματα με τη χρήση συναρτήσεων.
- Οι συναρτήσεις μπορεί να χρειάζεται να προκαλέσουν εξαιρέσεις που θα συλλαμβάνονται σε άλλες συναρτήσεις.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample8.cpp

`noexcept` και `throw()`

- Στη δήλωση μιας συνάρτησης μπορεί να προσδιοριστεί ότι ο συγκεκριμένος κώδικας δεν προκαλεί εξαιρέσεις. Αυτό γίνεται με τη δεσμευμένη λέξη `noexcept`.
- Αν μια συνάρτηση έχει δηλωθεί ως `noexcept` και προκαλεί εξαίρεση ή καλεί άλλη συνάρτηση που μπορεί να προκαλέσει εξαίρεση τότε δημιουργείται σφάλμα και καλείται η `std::terminate`.
- Το `noexcept` είναι ισοδύναμο με το `noexcept(true)` και με το `throw()`.
- Το `noexcept(false)` σηματοδοτεί ότι ο κώδικας μπορεί να προκαλέσει εξαιρέσεις και η προκαθορισμένη συμπεριφορά του είναι να μεταδώσει την εξαίρεση ένα επίπεδο παραπάνω (propagate), δηλαδή στη συνάρτηση που κάλεσε τον κώδικα που προκάλεσε την εξαίρεση.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect18/sample9.cpp

Κόστος εξαιρέσεων

- Οι εξαιρέσεις όταν συμβαίνουν είναι κατά πολύ περισσότερο χρονοβόρες σε σχέση με τους απλούς ελέγχους `if`.
- Συνεπώς, ή χρήση τους συνίσταται όταν η συνάρτηση από μόνη της δεν μπορεί να χειριστεί το συμβάν ή δεν είναι σχεδιαστικά σωστό να το κάνει καθώς η συγκεκριμένη αρμοδιότητα μπορεί να ανήκει σε κάποια άλλη κλάση.

Άσκηση #1: Exceptions

1. Γράψτε ένα πρόγραμμα που προσομοιώνει την ανάγνωση μιας τιμής από έναν «αισθητήρα», χρησιμοποιώντας εξαιρέσεις που προκαλούν ρίψη (throw) ακέραιων τιμών. Ειδικότερα, μια συνάρτηση με όνομα read_sensor(int x) να επιστρέφει την τιμή x, αλλά αν το x είναι αρνητικό να κάνει throw x. Στη main() καλέστε τη read_sensor με μια τιμή που θα δίνει ο χρήστης, χειριστείτε τις εξαιρέσεις με catch(int e) και εμφανίστε είτε το αποτέλεσμα είτε τον ακέραιο που έγινε throw.
2. Τροποποιήστε την άσκηση έτσι ώστε να ορίζετε μια custom κλάση εξαιρέσεων με όνομα SensorException και να γίνεται throw ένα αντικείμενο αυτής της κλάσης αντί για έναν απλό ακέραιο όταν η συνάρτηση read_sensor(int x) δέχεται ως όρισμα έναν αρνητικό αριθμό.

Άσκηση #1: Λύση (1/2)

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

int read_sensor(int x) {
  if (x < 0) throw x;
  return x;
}

int main() {
  int x;
  cout << "Enter sensor value: ";
  cin >> x;

  try {
    int value = read_sensor(x);
    cout << "Sensor reading: " << value << endl;
  } catch (int e) {
    cout << "Exception thrown with value: " << e << endl;
  }
 return 0;
}
```

```
Enter sensor value: 10
Sensor reading: 10
```

```
Enter sensor value: -5
Exception thrown with value: -5
```

Άσκηση #1: Λύση (2/2)

```cpp
#include <exception>
#include <iostream>
using namespace std;

class SensorException : public exception {
 private:
  int val;
 public:
  SensorException(int value) : val(value) {}
  const char* what() const noexcept override { return "Negative sensor value"; }
  int value() const { return val; }
};
int read_sensor(int x) {
  if (x < 0) throw SensorException(x);
  return x;
}
int main() {
  int x;
  cout << "Enter sensor value: ";cin >> x;
  try {
    int value = read_sensor(x);
    cout << "Sensor reading: " << value << endl;
  } catch (const SensorException& e) {
    cout << "Exception: " << e.what() << " (value = " << e.value() << ")"
         << endl;
  }
  return 0;
}
```

```
Enter sensor value: 10
Sensor reading: 10
```

```
Enter sensor value: -5
Exception thrown with value: -5
```

Άσκηση #2: Εξαιρέσεις τυπικής βιβλιοθήκης

- Γράψτε πρόγραμμα που να χειρίζεται εξαιρέσεις της τυπικής βιβλιοθήκης std::out_of_range. Ζητήστε από τον χρήστη έναν ακέραιο δείκτη και χρησιμοποιήστε τον για να προσπελάσετε ένα std::vector<int> πέντε στοιχείων μέσα σε try–catch, χρησιμοποιώντας τη μέθοδο at() ώστε να προκαλείται αυτόματα out_of_range σε περίπτωση μη έγκυρης πρόσβασης. Αν δεν προκύψει εξαίρεση, να εμφανίζετε την τιμή του στοιχείου, ενώ αν προκύψει, να συλλαμβάνετε την εξαίρεση με catch(const std::out_of_range& e) και να εμφανίζετε το μήνυμα που επιστρέφει η what().

Άσκηση #2: Λύση

```cpp
#include <iostream>
#include <stdexcept>
#include <vector>
using namespace std;

int main() {
  vector<int> v = {10, 20, 30, 40, 50};
  int index;
  cout << "Enter an index: ";
  cin >> index;
  try {
    int value = v.at(index);
    cout << "Value at index " << index << ": " << value << endl;
  } catch (const out_of_range& e) {
    cout << "Exception: " << e.what() << endl;
  }
  return 0;
}
```

```
Enter an index: 2
Value at index 2: 30
```

```
Enter an index: -1
Exception: vector
```

Άσκηση #3: Επιπλέον εξαιρέσεις τυπικής βιβλιοθήκης

- Γράψτε ένα πρόγραμμα που να επιδεικνύει τη χρήση των εξαιρέσεων std::invalid_argument και std::runtime_error. Το πρόγραμμα να ζητά από τον χρήστη δύο ακέραιους αριθμούς: τον αριθμό των βαθμών μιας εξέτασης και το πλήθος των σωστών απαντήσεων. Να υλοποιήσετε μια συνάρτηση compute_score(int total, int correct) η οποία να ελέγχει αν το total είναι μηδενικό ή αρνητικό και σε αυτή την περίπτωση να κάνει throw std::invalid_argument("total must be positive"). Στη συνέχεια, να ελέγχει αν το correct είναι μεγαλύτερο από το total και αν ναι, να κάνει throw std::runtime_error("correct answers exceed total questions"). Αν δεν προκύψει εξαίρεση, η συνάρτηση να επιστρέφει το ποσοστό επιτυχίας (correct/total) ως τιμή κινητής υποδιαστολής. Στη main() να καλέσετε τη συνάρτηση μέσα σε try-catch και να εμφανίσετε είτε το αποτέλεσμα είτε τα μηνύματα των εξαιρέσεων, χρησιμοποιώντας κατάλληλα catch(const std::invalid_argument&) και catch(const std::runtime_error&).

Άσκηση #3: Λύση

```cpp
#include <iostream>
#include <stdexcept>
using namespace std;

double compute_score(int total, int correct) {
  if (total <= 0) throw invalid_argument("total must be positive");
  if (correct > total)
    throw runtime_error("correct answers exceed total questions");
  return static_cast<double>(correct) / total;
}

int main() {
  int total, correct;
  cout << "Enter total number of questions: ";
  cin >> total;
  cout << "Enter number of correct answers: ";
  cin >> correct;
  try {
    double score = compute_score(total, correct);
    cout << "Score: " << score * 100 << "%\n";
  } catch (const invalid_argument& e) {
    cout << "Invalid argument: " << e.what() << endl;
  } catch (const runtime_error& e) {
    cout << "Runtime error: " << e.what() << endl;
  }
  return 0;
}
```

```
Enter total number of questions: -1
Enter number of correct answers: 10
Invalid argument: total must be positive
```

```
Enter total number of questions: 20
Enter number of correct answers: 21
Runtime error: correct answers exceed total questions
```

```
Enter total number of questions: 20
Enter number of correct answers: 15
Score: 75%
```

Ερωτήσεις σύνοψης

- Τι είναι και πως λειτουργούν τα `try-throw-catch` μπλοκς;
- Ποια είναι τα πλεονεκτήματα και ποια τα μειονεκτήματα των εξαιρέσεων;
- Τι μπορεί να γίνει `throw` σε έναν κώδικα;
- Ποιος είναι ο ρόλος του `catch(…)`, σε ποια θέση τοποθετείται και γιατί;
- Τι είναι η κλάση `std::exception`;
- Τι συμβαίνει αν μια εξαίρεση δεν συλλαμβάνεται (γίνεται `catch`);
- Ποιος είναι ο ρόλος της δεσμευμένης λέξης `noexcept`;
- Ποιος είναι ο ρόλος της εξαίρεσης `logic_error` ποιος της εξαίρεσης `runtime_error`;

Αναφορές

- http://www.cs.fsu.edu/~xyuan/cop3330/