Πέρασμα με τιμή και πέρασμα με αναφορά, οι δεσμευμένες λέξεις const και constexpr

#5

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Πανεπιστήμιο Ιωαννίνων (Άρτα)
Γκόγκος Χρήστος

Πέρασμα με τιμή (pass by value)

- Στη C++, όταν περνάμε μια μεταβλητή με τιμή, η συνάρτηση λαμβάνει ένα αντίγραφο της αρχικής μεταβλητής
- Τυχόν αλλαγές της μεταβλητής μέσα στη συνάρτηση δεν επηρεάζουν την αρχική μεταβλητή
- Αυτό έχει το πλεονέκτημα της ασφάλειας καθώς η αρχική μεταβλητή δεν τροποποιείται
- Από την άλλη μεριά αν η μεταβλητή είναι μεγάλου μεγέθους (π.χ., πίνακας ή δομή ή αντικείμενο) τότε δημιουργείται αντίγραφο που αυξάνει το χρόνο εκτέλεσης και τη μνήμη που καταναλώνεται

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

void f(int x) { x = x + 10; }

int main() {
  int a = 5;
  cout << "Before: " << a << endl;
  f(a);
  cout << "After: " << a << endl;
  return 0;
}
```

```
Before: 5
After: 5
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/pass_by_value.cpp

Πέρασμα με αναφορά (pass by reference)

- Με το πέρασμα με αναφορά, η συνάρτηση έχει άμεση πρόσβαση στην αρχική μεταβλητή μέσω αναφοράς
- Τυχόν αλλαγές στη συνάρτηση επηρεάζουν την αρχική μεταβλητή
- Είναι αποδοτικός τρόπος για πέρασμα μεγάλων δομών δεδομένων, αφού δεν δημιουργούνται αντίγραφα

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

void f(int& x) { x = x + 10; }

int main() {
  int a = 5;
  cout << "Before: " << a << endl;
  f(a);
  cout << "After: " << a << endl;
  return 0;
}
```

```
Before: 5
After: 15
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/pass_by_reference.cpp

Πέρασμα με αναφορά με τον τρόπο της C

- Στη C, δεν υπάρχει άμεσο πέρασμα με αναφορά αλλά χρησιμοποιούμε δείκτες για να περάσουμε τη διεύθυνση της μεταβλητής
- Στη C++, μπορεί επίσης να χρησιμοποιηθεί αυτός ο τρόπος, ως εναλλακτική του περάσματος με αναφορά με το &

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

void f(int* x) { *x = *x + 10; }

int main() {
  int a = 5;
  cout << "Before: " << a << endl;
  f(&a);
  cout << "After: " << a << endl;
  return 0;
}
```

```
Before: 5
After: 15
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/pass_by_reference2.cpp

Άσκηση #1: Εκφώνηση

- Να γραφεί πρόγραμμα που να επιδεικνύει τη διαφορά μεταξύ περάσματος παραμέτρων με τιμή (call by value) και με αναφορά (call by reference).
- Να οριστούν δύο συναρτήσεις που αυξάνουν μια ακέραια μεταβλητή κατά 1, η πρώτη λαμβάνοντας την παράμετρο με τιμή και η δεύτερη με αναφορά, και να εμφανίζεται στην οθόνη η τιμή της μεταβλητής πριν και μετά από κάθε κλήση συνάρτησης.

Άσκηση #1: Λύση

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

void incrementByValue(int x) { x++; }

void incrementByReference(int& x) { x++; }

int main() {
  int num = 10;

  cout << "Αρχική τιμή: " << num << endl;

  incrementByValue(num);
  cout << "Μετά το incrementByValue: " << num << endl;

  incrementByReference(num);
  cout << "Μετά το incrementByReference: " << num << endl;

  return 0;
}
```

```
Αρχική τιμή: 10
Μετά το incrementByValue: 10
Μετά το incrementByReference: 11
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/exercise1.cpp

const

- Γενικά, η δεσμευμένη λέξη `const` εφαρμόζεται σε αναγνωριστικά προκειμένου να δηλώσει την πρόθεσή του ότι το αναγνωριστικό δεν θα πρέπει να χρησιμοποιείται για να αλλάξει δεδομένα στα οποία αναφέρεται.
- Ο μεταγλωττιστής επιβάλλει την πρόθεση του προγραμματιστή, για παράδειγμα (t1.cpp):

```
int main()
{
        const int x = 5;
        x = 2;
}
```

```
g++ t1.cpp
t1.cpp:4:5: error: cannot assign to variable 'x' with const-qualified type 'const int'
 x = 2;
 ~ ^
t1.cpp:3:13: note: variable 'x' declared const here
 const int x = 5;
 ~~~~~~~~~~^~~~~
1 error generated.
```

- Στο παραπάνω παράδειγμα, το x είναι ένα αναγνωριστικό που αναφέρεται σε ακέραια δεδομένα στη μνήμη και ο μεταγλωττιστής **επιβάλλει** ότι το x δεν θα πρέπει να χρησιμοποιείται με οποιοδήποτε τρόπο που θα οδηγούσε σε αλλαγή των δεδομένων.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/t1.cpp

const

- Σε ένα άλλο παράδειγμα (t2.cpp):

```
int main()
 {
        const int x = 5;
        int *x_ptr;
        x_ptr = & x;
}
```

```
g++ t2.cpp
t2.cpp:6:11: error: assigning to 'int *' from incompatible type 'const int *'
  x_ptr = & x;
        ^~~
1 error generated.
```

- Στο παραπάνω παράδειγμα, ο μεταγλωττιστής δεν επιτρέπει στο δείκτη `x_ptr` να δείξει προς τη διεύθυνση του `x` (που είναι τύπου `const int`). Αυτό συμβαίνει διότι ο `x_ptr` θα μπορούσε να χρησιμοποιηθεί για να αλλάξει τα δεδομένα που είναι αποθηκευμένα στο `x`.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/t2.cpp

Επισκόπηση: const παράμετροι

- Στη C++ υπάρχουν δύο τρόποι με τους οποίους μπορούν να περάσουν ορίσματα στις συναρτήσεις:
  - **Πέρασμα με τιμή** (call by value): Ένα αντίγραφο του ορίσματος δημιουργείται και η συνάρτηση επενεργεί σε αυτό το αντίγραφο.
  - **Πέρασμα με αναφορά** (call by reference): Δεν δημιουργείται αντίγραφο, η παράμετρος της συνάρτησης είναι ένα αναγνωριστικό που αναφέρεται στα ίδια δεδομένα με το όρισμα (η παράμετρος λειτουργεί ως ένα **ψευδώνυμο** για το όρισμα).
- Η μέθοδος που χρησιμοποιείται για το πέρασμα των ορισμάτων υποδηλώνεται από τις παραμέτρους της συνάρτησης:

```
void foo(int x);  // το x περνά με τιμή
void foo(int &x); // το x περνά με αναφορά
```

- Ποια είναι η κύρια διαφορά ανάμεσα στους δύο αυτούς μηχανισμούς περάσματος παραμέτρων (δείτε το t3.cpp);

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/t3.cpp

Επισκόπηση: const παράμετροι

- Ο προγραμματιστής μπορεί να προσθέτει τη δεσμευμένη λέξη `const` και στους δύο μηχανισμούς περάσματος παραμέτρων (t4.cpp):

```
void foo(const int x);  /* το x είναι μια παράμετρος εισόδου
(μόνο για ανάγνωση), δεν μπορεί να τροποποιηθεί εντός της foo */
void foo(const int &x); /* το x είναι μια παράμετρος εισόδου
(μόνο για ανάγνωση), δεν μπορεί να τροποποιηθεί εντός της foo */
```

- Ποια είναι η διαφορά ανάμεσα στους δύο αυτούς μηχανισμούς περάσματος παραμέτρων;
  - Το πέρασμα με τιμή πρέπει να κατασκευάσει ένα αντίγραφο.
  - Το πέρασμα με αναφορά δεν κατασκευάζει αντίγραφο (απλά περνά την αναφορά της πραγματικής παραμέτρου στο υποπρόγραμμα)
    - Η αναφορά μιας μεταβλητής είναι απλά ένας δείκτης προς τη μεταβλητή (8 bytes)
    - Το πέρασμα με αναφορά (`const` ή όχι) είναι πολύ χρήσιμο όταν η παράμετρος είναι μια μεγάλη δομή δεδομένων έτσι ώστε να μειώσει την επιβάρυνση που θα δημιουργούταν από την κατασκευή ενός αντιγράφου.
- Δείτε το t4.cpp.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/t4.cpp

Πέρασμα αντικειμένων με const αναφορά

- Τα αντικείμενα μπορούν επίσης να περάσουν με `const` αναφορά έτσι ώστε να αποφευχθεί η επιβάρυνση αντιγραφής:

```
friend Fraction Add(const Fraction& f1, const Fraction& f2);
```

- Όπως και με τους άλλους τύπους δεδομένων, ο μεταγλωττιστής διασφαλίζει ότι ένα αντικείμενο που περνά με `const` αναφορά δεν θα χρησιμοποιηθεί με τέτοιο τρόπο που θα μπορούσε να αλλάξει τα μέλη δεδομένα του.

Άσκηση #2: εκφώνηση

- Δημιουργήστε μια κλάση Book που αναπαριστά ένα βιβλίο με τίτλο, συγγραφέα, περιεχόμενο και αριθμό σελίδων (όλα ως strings εκτός από τις σελίδες). Στη συνέχεια, υλοποιήστε μια συνάρτηση printBookInfo με τρεις διαφορετικούς τρόπους: (1) πέρασμα με αντιγραφή Book book, (2) πέρασμα με αναφορά Book& book, και (3) πέρασμα με const αναφορά const Book& book. Δημιουργήστε ένα Book αντικείμενο με μεγάλο string για τον περιεχόμενο (π.χ. 10.000 χαρακτήρες) και καλέστε κάθε έκδοση 1.000.000 φορές καταμετρώντας τον χρόνο εκτέλεσης.
- Ποια έκδοση είναι ταχύτερη και γιατί;
- Γιατί η const& αναμένεται να είναι ταχύτερη από την απλή &;
- Τι συμβαίνει αν προσπαθήσετε να τροποποιήσετε το αντικείμενο με const&;

Άσκηση #2: Λύση

```cpp
#include <chrono>
#include <iostream>
#include <string>
using namespace std;
using namespace std::chrono;

class Book {
 public:
  string title, author, content;
  int pages;
  Book(string t, string a, string c, int p)
      : title(t), author(a), content(c), pages(p) {}
};

void printBookInfo(Book book) {
  int dummy = book.pages;
}
void printBookInfoRef(Book& book) {
  int dummy = book.pages;
}
void printBookInfoConstRef(const Book& book) {
  int dummy = book.pages;
}
```

```cpp
int main() {
string largeContent(10000, 'x');
Book b("Title", "Author", largeContent, 123);


auto start1 = high_resolution_clock::now();
for (int i = 0; i < 1000000; ++i) printBookInfo(b);
auto end1 = high_resolution_clock::now();
cout << "Pass by value: "
<< duration_cast<milliseconds>(end1 - start1).count() << " ms\n";


auto start2 = high_resolution_clock::now();
for (int i = 0; i < 1000000; ++i) printBookInfoRef(b);
auto end2 = high_resolution_clock::now();
cout << "Pass by reference: "
<< duration_cast<milliseconds>(end2 - start2).count() << " ms\n";


auto start3 = high_resolution_clock::now();
for (int i = 0; i < 1000000; ++i) printBookInfoConstRef(b);
auto end3 = high_resolution_clock::now();
cout << "Pass by const reference: "
<< duration_cast<milliseconds>(end3 - start3).count() << " ms\n";


return 0;
}
```

```
Pass by value: 168 ms
Pass by reference: 1 ms
Pass by const reference: 1 ms
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/exercise2.cpp

const συνάρτηση μέλος

- Οποιαδήποτε κλήση σε μια συνάρτηση μέλος διαθέτει το «καλών αντικείμενο» (δηλαδή η συνάρτηση μπορεί να έχει πρόσβαση στο καλών αντικείμενο)

```
Fraction f1;   // ένα αντικείμενο Fraction
f1.Evaluate(); // το f1 είναι το καλών αντικείμενο
```

- Καθώς η συνάρτηση μέλος έχει πρόσβαση στα δεδομένα του καλούντος αντικειμένου, μπορεί να θέλουμε να διασφαλίσουμε ότι το καλών αντικείμενο δεν πρόκειται ποτέ να αλλάξει από τη συνάρτηση μέλος.
- Τέτοιες συναρτήσεις ονομάζονται **`const` συναρτήσεις μέλη** και υποδηλώνονται από τη χρήση της δεσμευμένης λέξης const μετά τη δήλωση της συνάρτησης καθώς **και** στον ορισμό της συνάρτησης.
- Δείτε το t5.cpp.

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/t5.cpp

const αντικείμενα

- const μεταβλητές είναι εκείνες οι μεταβλητές που λαμβάνουν μόνο μια τιμή (αυτή με την οποία αρχικοποιήθηκαν) και που δεν μπορεί να αλλάξει.

```
const int SIZE = 10;
const double PI = 3.1415;
```

- Τα αντικείμενα μπορούν να δηλωθούν με παρόμοιο τρόπο ως const. Ο κατασκευαστής αρχικοποιεί τα const αντικείμενα, αλλά μετά από αυτό, τα μέλη δεδομένα του αντικειμένου δεν θα μπορούν να αλλάξουν.

```
const Fraction ZERO; // το κλάσμα ορίζεται σε 0/1 και δεν αλλάζει
const Fraction FIXED(3,4);// το κλάσμα ορίζεται σε 3/4 και δεν αλλάζει
```

- Για να διασφαλιστεί ότι ένα αντικείμενο δεν μπορεί να αλλάξει, ο μεταγλωττιστής επιβάλλει ότι ένα const αντικείμενο μπορεί να καλεί μόνο const συναρτήσεις μέλη.
- Δείτε το παράδειγμα const_fraction.

https://github.com/chgogos/oop/tree/master/various/COP3330/lect5/const_fraction

const μέλη δεδομένων

- Τα μέλη δεδομένων μιας κλάσης μπορούν επίσης να δηλωθούν ως `const`, αλλά υπάρχουν ορισμένοι συντακτικοί κανόνες που πρέπει να τηρούνται.
- Γνωρίζουμε ότι όταν μια μεταβλητή δηλώνεται με το `const` σε ένα μπλοκ κώδικα, θα πρέπει να αρχικοποιείται στην ίδια γραμμή:

```
const int SIZE = 10;
```

- Το ίδιο μπορεί να συμβεί και με τις const μεταβλητές μέλη δεδομένων (από τη C++11 και μετά).
- Ωστόσο, η απόπειρα να αναθέσουμε τιμή σε `const` μεταβλητή μέλος μέσα σε έναν κατασκευαστή (ή σε οποιαδήποτε άλλη συνάρτηση μέλος) δεν θα λειτουργήσει.

Λίστα αρχικοποίησης

- Μπορούμε να χρησιμοποιήσουμε μια ειδική περιοχή του κατασκευαστή που ονομάζεται **λίστα αρχικοποίησης (initializer list)** έτσι ώστε να ξεπεράσουμε το πρόβλημα αρχικοποίησης const μελών αντικειμένων στον κατασκευαστή.
- Οι λίστες αρχικοποίησης έχουν την ακόλουθη μορφή:

```
classname::classname(int p1, int p2): const_member_var1(p1), member_var2(p2)
{
  // κώδικας
}
```

- Η παραπάνω λίστα αρχικοποίησης θα θέσει το `const_member_var1` και το `member_var2` στις τιμές που περνάνε στον κατασκευαστή ως `p1` και `p2` αντίστοιχα.

Άσκηση #3: Εκφώνηση

- Να γραφεί πρόγραμμα που να επιδεικνύει τη χρήση του const σε διάφορα σημεία στα οποία μπορεί να χρησιμοποιηθεί.
- Να οριστεί μια κλάση Circle με ένα const μέλος δεδομένων pi με τιμή 3.14, ένα μη-const μέλος radius (ακτίνα κύκλου), έναν κατασκευαστή, έναν setter για την ακτίνα και μια const συνάρτηση μέλος area() που επιστρέφει τον υπολογισμό του εμβαδού του κύκλου.
- Στη main() να δημιουργηθεί ένα const αντικείμενο της κλάσης Circle και να καλείται η συνάρτηση area(), δείχνοντας πως το const περιορίζει τροποποιήσεις.

Άσκηση #3: Λύση

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

class Circle {
  const double pi;
  double radius;

 public:
  Circle(double r) : pi(3.14), radius(r) {}

  double area() const { return pi * radius * radius; }

  void setRadius(double r) { radius = r; }
};

int main() {
  const Circle c(5.0);
  cout << "Area: " << c.area() << endl;

  // c.setRadius(10.0);  // Σφάλμα

  return 0;
}
```

```
Area: 78.5
```

https://github.com/chgogos/oop/blob/master/various/COP3330/lect5/exercise3.cpp

constrexpr

- Η χρήση του constexpr κατά τη δήλωση μιας μεταβλητής υποδηλώνει ότι η τιμή που της ανατίθεται πρέπει να είναι γνωστή και να μπορεί να υπολογιστεί κατά το χρόνο μεταγλώττισης
- Το constexpr μπορεί επίσης να χρησιμοποιηθεί κατά τη δήλωση μιας συνάρτησης υποδηλώνοντας ότι η συνάρτηση μπορεί να αποτιμηθεί κατά το χρόνο μεταγλώττισης (π.χ. η συνάρτηση δεν δεσμεύει δυναμικά μνήμη, δεν έχει εντολές εισόδου και γενικά δεν χρησιμοποιεί κάτι που παράγεται κατά το χρόνο εκτέλεσης)
- Πλεονεκτήματα του constexpr είναι:
  - Υψηλότερη ταχύτητα λόγω υπολογισμών κατά το χρόνο μεταγλώττισης
  - Μεγαλύτερη ασφάλεια λόγω επαλήθευσης ορθότητας κώδικα κατά το χρόνο μεταγλώττισης

```cpp
#include <iostream>
using namespace std;
using ull = unsigned long long;

constexpr ull factorial(int n) {
    return (n <= 1) ? 1 : n * factorial(n - 1);
}

int main() {
  // υπολογισμός κατά το χρόνο μεταγλώττισης
  constexpr ull f5 = factorial(5);
  cout << f5 << endl;

  int x = 5;
  // υπολογισμός κατά το χρόνο εκτέλεσης
  ull fx = factorial(x);
  cout << fx << endl;
  return 0;
}
```

```
120
120
```

Ερωτήσεις σύνοψης

- Περιγράψτε το πέρασμα με τιμή και το πέρασμα με αναφορά.
- Γιατί χρησιμοποιούμε το πέρασμα με αναφορά;
- Σε τι αναφέρεται ο όρος «καλών αντικείμενο»;
- Τι είναι μια const συνάρτηση μέλος;
- Τι είναι ένα const αντικείμενο και πως το αρχικοποιούμε;
- Τι είναι const μέλος δεδομένων και πως το αρχικοποιούμε;
- Σε τι επίπεδο (μεταγλώττισης ή εκτέλεσης) επιβάλλεται το const;
- Ποια είναι η διαφορά constexpr και const;

Απαντήσεις στις ερωτήσεις σύνοψης

- Περιγράψτε το πέρασμα με τιμή και το πέρασμα με αναφορά.
  - Στο πέρασμα με τιμή δουλεύουμε στη συνάρτηση με ένα αντίγραφο, ενώ στο πέρασμα με αναφορά δουλεύουμε με την ίδια τη μεταβλητή.
- Γιατί χρησιμοποιούμε το πέρασμα με αναφορά;
  - Χρησιμοποιούμε το πέρασμα με αναφορά για να μπορούμε να τροποποιούμε τις μεταβλητές που περνούν ως ορίσματα στη συνάρτηση, μέσα στη συνάρτηση, χωρίς να δημιουργούνται αντίγραφα.
- Σε τι αναφέρεται ο όρος «καλών αντικείμενο»;
  - Ο όρος «καλών αντικείμενο» αναφέρεται στο αντικείμενο που καλεί μια μέθοδο, δηλαδή στο αντικείμενο στο οποίο εφαρμόζεται η συγκεκριμένη συνάρτηση μέλους.
- Τι είναι μια const συνάρτηση μέλος;
  - Μια const συνάρτηση μέλος είναι μια συνάρτηση που δεν επιτρέπεται να τροποποιήσει τα δεδομένα μέλη του αντικειμένου στο οποίο ανήκει και δηλώνεται με τη λέξη-κλειδί const μετά την επικεφαλίδα της.
- Τι είναι ένα const αντικείμενο και πως το αρχικοποιούμε;
  - Ένα const αντικείμενο είναι ένα αντικείμενο του οποίου οι τιμές δεν μπορούν να τροποποιηθούν μετά την αρχικοποίησή του. Δηλώνεται με τη λέξη-κλειδί const και πρέπει να αρχικοποιηθεί τη στιγμή της δημιουργίας του.
- Τι είναι const μέλος δεδομένων και πως το αρχικοποιούμε;
  - Ένα const μέλος δεδομένων είναι μια μεταβλητή μέλους μιας κλάσης της οποίας η τιμή δεν μπορεί να αλλάξει μετά την αρχικοποίησή της. Επειδή δεν μπορεί να της αποδοθεί τιμή μέσα στο σώμα του constructor, αρχικοποιείται μέσω της λίστας αρχικοποίησης του constructor.
- Σε τι επίπεδο (μεταγλώττισης ή εκτέλεσης) επιβάλλεται το const;
  - Το const επιβάλλεται σε επίπεδο μεταγλώττισης από τον compiler, ο οποίος ελέγχει ότι δεν γίνονται μη επιτρεπτές τροποποιήσεις των const αντικειμένων, μελών ή παραμέτρων, και παράγει σφάλμα αν επιχειρηθεί αλλαγή.
- Ποια είναι η διαφορά constexpr και const;
  - Το const δηλώνει τιμή που δεν αλλάζει, ενώ το constexpr απαιτεί η τιμή να είναι γνωστή και υπολογίσιμη κατά τη μεταγλώττιση.

Αναφορές

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