Προγραμματισμός Υπολογιστών με C++

Συναρτησιακός Προγραμματισμός

Γεώργιος Παπαϊωάννου (2015-22)
gepap@aueb.gr

Περιεχόμενο Παρουσίασης

- Περιγραφή:
  - Ο τύπος «συνάρτηση»
  - Lambda functions
  - Η βιβιοθήκη <algorithm>

- Τελευταία ενημέρωση: Δεκ 2022

Μέθοδοι ως Συναρτήσεις

- Μπορώ να καλέσω μια μέθοδο σαν μια απλή συνάρτηση;
  - Κανονικά όχι, εκτός και αν η μέθοδος είναι στατική
- Η γλώσσα όμως (C++11), μας επιτρέπει να δούμε μια μέθοδο ενός υπαρκτού στιγμιότυπου ως συνάρτηση με τη συνάρτηση μετατροπής **bind**

Μέθοδοι ως Ελεύθερες Συναρτήσεις

```
#include <iostream>
#include <functional>
struct int_holder {
    int value;
    int triple() { return value*3;}
};

int main () {
    int_holder five {5};
    std::function<int()> triple =
        std::bind (&int_holder::triple, five);
    cout << triple();
    thread th = thread (triple);
}
```

Αντικείμενο – περίβλημα (wrapper) για συναρτήσεις. Οτιδήποτε μπορεί κάποια στιγμή να κληθεί, μπορεί να μετατραπεί σε αυτόν τον τύπο.

Διεύθυνση στην οποία βρίσκεται η αρχή της υλοποίησης της μεθόδου

Μέθοδοι ως Ελεύθερες Συναρτήσεις – Callbacks

- Έστω ότι έχουμε μια κλάση MoviePlayer από την οποία θα θέλαμε να καλούμε για στιγμιότυπα αυτής τη μέθοδο controlPlayer, όταν πατιέται ένα κουμπί σε ένα user interface:

```
class MoviePlayer
{
...

public:
   void controlPlayer(int evt, void * data)
   {
        switch (evt)
        {
        case 1: play(data); break;
        case 2: stop();
        }
   }
};
```

- Ανάλογα με το ποιο είναι το αναγνωριστικό του evt, θα θέλαμε να αντιδρά διαφορετικά, όπως φαίνεται στον παραπάνω κώδικα

Μέθοδοι ως Ελεύθερες Συναρτήσεις – Callbacks

- Από την άλλη πλευρά, ένα γραφικό στοιχείο, όπως ένα κουμπί, δε θέλουμε να πρέπει να γνωρίζει ότι ενεργοποιεί κάποιο αντικείμενο τύπου MoviePlayer, καθώς κάτι τέτοιο καταλύει οποιονδήποτε γενικό σχεδιασμό.

- Για μια κλάση που θέλει να απαντά σε γεγονότα καλώντας κώδικα έξω από το scope αυτής (που μπορεί μάλιστα να αλλάζει δυναμικά το ποιος θα είναι), η τυπική και γενική αντιμετώπιση είναι μέσω callback functions

- Μια τέτοια συνάρτηση (callback) αποθηκεύεται ως δεδομένο με τη μορφή:
  - Δείκτη σε συνάρτηση, ή καλύτερα
  - function<> object

Μέθοδοι ως Ελεύθερες Συναρτήσεις – Callbacks

- Έτσι, στο παράδειγμά μας:

```
class Widget
{
protected:
    bool action_occured = false;
    std::function<void(int, void*)> action_callback;

public:
    void addActionCallback(std::function<void(int, void*)> cb)
    {
        action_callback = cb;
    }

    void update()
    {
        ...
        if (action_occured)
        {
            if (action_callback)
                action_callback(0, nullptr);
        }
    }
};
```

Μέθοδοι ως Ελεύθερες Συναρτήσεις – Callbacks

- Έτσι, στο παράδειγμά μας:

```
class Widget
{
protected:
    bool action_occured = false;
    std::function<void(int, void*)> action_callback;

public:
    void addActionCallback(std::function<void(int, void*)> cb)
    {
        action_callback = cb;
    }

void update()
{
    ...
    if (action_occured)
    {
        if (action_callback)
        action_callback(0, this);
    }
}
};
```

Προσοχή: αυτά τα ορίσματα, αν δεν κάνουμε εμείς ειδική μεταχείριση (βλ. συνέχεια), δε χρησιμοποιούνται!

Μέθοδοι ως Ελεύθερες Συναρτήσεις – Callbacks

- Ας συνδέσουμε τώρα τις callback functions:

```
MoviePlayer player;
Widget button_play;
Widget button_stop;
button_play.addActionCallback(
    std::bind(&MoviePlayer::controlPlayer, &player, 1, 0)
);

button_stop.addActionCallback(
    std::bind(&MoviePlayer::controlPlayer, &player, 2, 0)
);
```

1ο όρισμα μεθόδου

2ο όρισμα μεθόδου

(στατική) διεύθυνση μεθόδου

διεύθυνση στιγμιότυπου του οποίου η μέθοδος θα κληθεί

Προσοχή: αυτά τα στατικά περασμένα ορίσματα υπερισχύουν έναντι των όποιων ορισμάτων περνάμε κατά την κλήση της bound μεθόδου

Μέθοδοι ως Ελεύθερες Συναρτήσεις – Callbacks

- Ας συνδέσουμε τώρα τις callback functions:

```
MoviePlayer player;
Widget button_play;
Widget button_stop;
button_play.addActionCallback(
   std::bind(&MoviePlayer::controlPlayer, &player, 1, std::placeholders::_2)
);

button_stop.addActionCallback(
   std::bind(&MoviePlayer::controlPlayer, &player, 2, std::placeholders::_2)
);
```

Τώρα πλέον λέμε ότι σαν δεύτερο όρισμα της bound function να χρησιμοποιείται το δεύτερο όρισμα που περνάμε κατά την κλήση της

Μέθοδοι ως Ελεύθερες Συναρτήσεις – Callbacks

- Δηλαδή πλέον η callback function χρησιμοποιεί πράγματι το δεύτερο όρισμα
- Που της περνάμε run-time.

```
class Widget
{
protected:
    bool action_occured = false;
    std::function<void(int, void*)> action_callback;

public:
    void addActionCallback(std::function<void(int, void*)> cb)
    {
        action_callback = cb;
    }

void update()
{
    ...
    if (action_occured)
    {
        if (action_callback)
        action_callback(0, this);
    }
}
};
```

Καλείται η action_callback με δεύτερο όρισμα το this (λόγω std::placeholders::_2)

Συνεχίζει να αγνοείται το πρώτο όρισμα εδώ. Υπερισχύει η τιμή που δώσαμε κατά το binding

Βιβλιοθήκη Αλγορίθμων για Συλλογές

- Η C++ μας παρέχει έτοιμες λειτουργίες για να εφαρμόσουμε μαζικά αλγορίθμους πάνω σε συλλογές
- Βρίσκονται στο header algorithm
- Π.χ.:
  - sort και binary_search
  - for_each
  - find_if, any_of, none_of, count, count_if
  - generate
  - …

Παράδειγμα: any_of

```
bool negative(int n) { return n < 0; }


int main() {
    std::array<int, 7> foo = { 0, 1, -1, 3, -3, 5, -5 };
    if (std::any_of(foo.begin(), foo.end(), negative))
        std::cout << "Found negative elements.\n";
}
```

array: wrapper class για έναν στατικό πίνακα, ώστε να διαθέτει:
A) iterator
B) size() μέθοδο

Παράδειγμα: transform

```
int negate(int & n) { return n*-1; }


int main() {
  std::array<int, 7> foo = { 0, 1, -1, 3, -3, 5, -5 };
  std::transform(foo.begin(), foo.end(), foo.begin(), negate);


  for (auto o : foo)
    std::cout << o << " ";
}
```

Lambda Functions

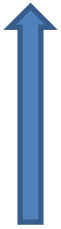
- Σε όλες τις παραπάνω περιπτώσεις, για να επενεργήσουμε πάνω σε κάποια δεδομένα, χρειαζόταν να δημιουργήσουμε πρώτα μια συνάρτηση ή ένα function object (ή εναλλακτικά ένα αντικείμενο με τον τελεστή () υλοποιημένο) - Callables
- Πολλές φορές δε θέλουμε να ορίσουμε και δημιουργήσουμε χωριστά ένα Callable αν πρόκειται να χρησιμοποιηθεί μόνο σε ένα σημείο
  - Μπορούμε να κατασκευάσουμε επί τόπου μια ανώνυμη συνάρτηση, ενδεχομένως μαζί με το περιβάλλον της (μεταβλητές ή άλλες συναρτήσεις) → **Lambda Function**

Σύνταξη Lambda Functions

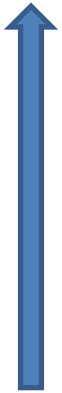
[a,b,&c] : Βλέπει τις μεταβλητές a και b κατά τιμή και την c με αναφορά
[] : Δε χρησιμοποιεί καμία μεταβλητή του περιβάλλοντος
[=] : Όλες τις μεταβλητές του περιβάλλοντος κατά τιμή
[&] : Όλες τις μεταβλητές του περιβάλλοντος με αναφορά

[*capture-list* **] (** *params* **) -> ** *ret* **{** *body* **}**

Παράμετροι συνάρτησης



Τύπος επιστροφής



Σώμα συνάρτησης

Lambda Functions – Παράδειγμα 1

```
int main() {
std::array<int, 7> foo = { 0, 1, -1, 3, -3, 5, -5 };



std::transform(foo.begin(), foo.end(), foo.begin(), [](int x)  {return x*=-1; });

for (auto o : foo)
std::cout << o << " ";

getchar();
}
```

Ανώνυμη συνάρτηση

-> int

[...]: Προσδιορίζει πώς χρησιμοποιεί η ανώνυμη συνάρτηση τις μεταβλητές του περιβάλλοντος κλήσης. Εδώ δεν τις χρειάζεται

Στις περισσότερες περιπτώσεις ο compiler μπορεί να εξάγει τον τύπο επιστροφής χωρίς να τον δώσουμε ρητά

Lambda Functions – Παράδειγμα 2

```
int main() {
    std::vector<int> some_list;
    const int multiplier = 2;
    int total = 0;
    for (int i = 0; i<5; i++) some_list.push_back(i);

    std::for_each(begin(some_list), end(some_list),
        [&total, multiplier](int x) {total += x*multiplier;} );

    std::cout << "total = " << total;
    getchar();
}
```

[…]: Χρησιμοποιεί την total με αναφορά (την αλλάζει) και την multiplier με αντιγραφή (δεν την αλλάζει)