ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ–ΗΥ352

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2021

ΔΙΔΑΣΚΩΝ: ΑΝΤΩΝΙΟΣ ΣΑΒΒΙΔΗΣ

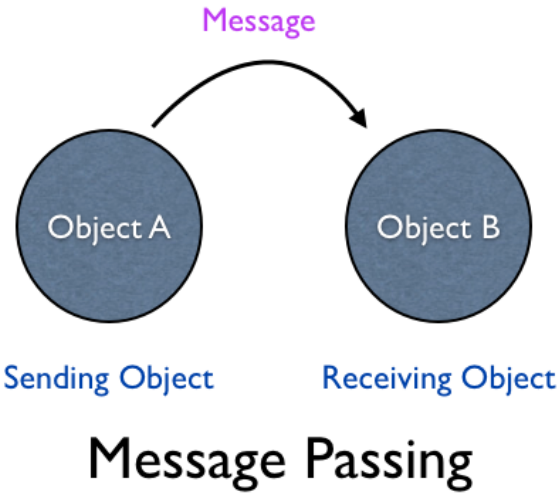
ΕΡΓΑΣΙΑ (ομάδες μέχρι δύο άτομα)

Ανάθεση: 13 Δεκεμβρίου 2021

Παράδοση: -

Θέμα – Κατασκευή γλώσσας για Message Passing

Θέμα της εργασίας είναι η κατασκευή μιας γλώσσας ειδικού σκοπού για τον ορισμό αντικειμένων που δέχονται μηνύματα προς εκτέλεση από άλλα αντικείμενα. Ένα μήνυμα προς ένα αντικείμενο είναι μια "αίτηση" για την εκτέλεση κάποιας διαδικασίας από αυτό.



Όπως φαίνεται στο παραπάνω σχήμα η επικοινωνία μέσω μηνυμάτων θα γίνεται μεταξύ ενος sending object A και ενός receiving object B. Τα objects μπορούν να αποθηκεύουν πεδία με δεδομένα και συναρτήσεις που εκτελούν λειτουργίες. Το receiving object B προσφέρει ένα τρόπο επεξεργασίας των δεδομένων και κλήσης των διαδικασιών που του αποστέλονται απο ένα sending object A.

Για να πετύχετε αυτή τη λειτουργικότητα πρέπει να δημιουργήσετε ένα σύνολο απο keywords που παρέχουν συγκεκριμμένο functionality (περιγράφονται παρακάτω) και καθορίζουν την δομή της γλώσσας.

Η γλώσσα που θα κατασκευάσετε θα γίνεται compiled ως C++ οπότε θα χρειαστείτε ένα ή περισσότερα header files με κατάλληλους ορισμούς ώστε το πρόγραμμά σας που είναι γραμμένο στη γλώσσα ειδικού σκοπού να αντιστοιχεί σε valid C++ κώδικα και να κάνει compile σωστά. Όπως και στη C++ τα whitespaces αγνοούνται.

Ένα πρόγραμμα θα έχει πάντα την εξής μορφή:

```
#include <MSGlang.h>

int main() {
   (τα definitions γίνονται με οποιαδήποτε σειρά αλλά πριν το pass του message)
  object definition
  message definition
  pass message to object
}
```

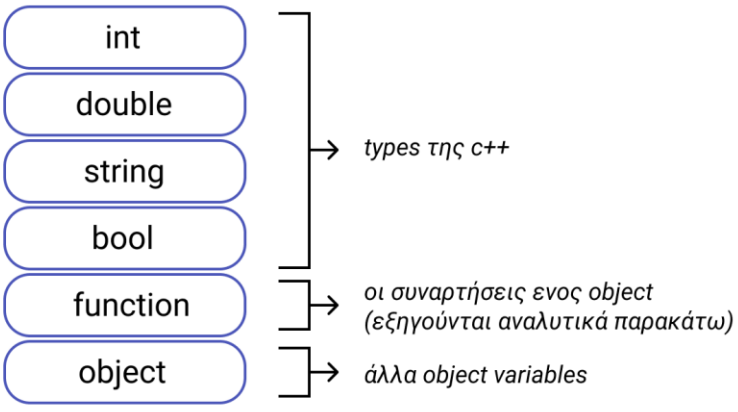
Στοιχεία της γλώσσας

Ορισμός Object variable

Τα objects μπορουν να δηλωθούν ως μεταβλητές τις γλώσσας ακολουθώντας το παρακάτω συντακτικό:

```
let name1 = object;
let name2 = object[ values val1, val2, val3... ];
let name3 = object[
   key(id1) = val1,
   key(id2) = val2,
   key(id3)= lambda{body1},
   func(id4){body2} ...
];
```

- Το ***name*** είναι το όνομα της μεταβλητής του object και ακολουθεί όλους του κανόνες ονομασίας μεταβλητών της C++.
- Το **let** keyword αντιπροσωπεύει οποιοδήποτε τύπο υποστηρίζει η c++ καθώς και user defined τύπους και ακολουθείται πάντοτε με initializer.
- Το **key** keyword χρησιμοποιείται για την δήλωση μιας μεταβλητής εντός του object την οποία κάνουμε access μέσω του id που περνάμε στις παρενθέσεις.
- Το ***val$_i$*** είναι το value του κάθε πεδίου του object μπορεί να πάρει τις ακόλουθες τιμές:



- Το ***id$_i$*** είναι **string**.
- Το **values** keyword ομαδοποιεί κάθε argument στα δεξιά του διαχωρισμένο με κομμα σε ένα array απο values.Τα values γίνονται προσβάσιμα με indexing όπως σε ένα array με τη διαφορά οτι είναι **strings** (δηλ. Indexed με “0”,”1”,… και όχι 0,1,… όπως στη C++.

- Τα objects θα πρέπει να μπορούν να εκτυπωθούν με την ακόλουθη μορφή. (Προσοχη! Τα πεδία που είναι τύπου συνάρτησης εκτυπώνωνται ως method)

  object [“id$_1$”:value$_1$, … , “id$_n$”: method];

- Ένα object μπορεί να έχει όσα πεδία θέλουμε.
- Το `none` keyword αντιστοιχεί στο κενό value. π.χ. `return none;` Επιστρέφει μια κενή μεταβλητή.

Παραδείγματα δήλωσης μεταβλητών object:

```
let o1 = object [
    key("x") = -1,              // μεταβλητή με id = “x” και τιμή -1
    key("y") = -2,
    func("mid") { return (self(x) + self(y)) * 0.5f; },
    func("+") {                 // μεταβλητή ως συνάρτηση με id = “+”
        return object [         // δήλωση προσωρινού object που θα επιστραφεί
            key("x") = self(x) + arg(x),
            key("y") = self(y) + arg(y)
        ];
    }
];

let o2 = object;  // άδειο object

let o3 = object [ values 1, "2", true, -3.14 ];  // object ως array απο values, indexed αυτόματα με "0", "1" , ...

std::cout << o3;  // τυπώνει: object [ “0”:1 , “1”:“2” , “2”:true , “3”:-3.14 ]
```

Πρόσβαση και επεξεργασία πεδίων του object

Τα objects θα πρέπει να είναι προσβάσιμα απο οποιοδήποτε σημείο του προγράμματος. Μπορούμε να έχουμε πρόσβαση στα πεδία τους μέσα απο το id τους και να αλλάξουμε την τιμή τους, όπως φένεται παρακάτω.

```
obj_name[id] = value; // το id πρέπει να είναι string

o1["x"];          // επιστρέφει το πεδίο με id “x” απο το object με όνομα o1 (δηλ. -1)
o1["x"] = 0;      // αλλάζει την τιμή του πεδίου “x” σε 0
o3["2"] = false; // αλλάζει την τιμή του πεδίου “2” σε false στο object o3
```

Επιπλέον, πρέπει να ορίσετε τα παρακάτω keywords που προσφέρουν τη δυνατότητα να αναφερθείτε σε υπάρχον object και να πάρετε είσοδο απο το χρήστη ως τιμή για πεδίο ενος object.

- **input:** Το input χρησιμοποιείται με το συντακτικό **input(**msg**)**, όπου το msg είναι ένα string. Είναι μία συνάρτηση που τυπώνει στην κονσόλα το msg, περιμένει είσοδο από τον χρήστη και μόλις το διαβάσει, το επιστρέφει. Η είσοδος που θα διαβαστεί μπορεί να έχει τύπο **double**, **int**, **bool** ή **string**. π.χ.:

```
o1["x"] = input("x:"); // Θα εκτυπωθεί το μήνυμα "x:" στην κονσόλα και θα περιμένει input
απο τον χρήστη επιστρέφοντας το. Έπειτα το input αποθηκεύεται στο πεδίο «χ» του object o1
```

- **ref:** Παίρνει ως όρισμα το όνομα μιας μεταβλητής object και μας δίνει ένα reference σε ολόκληρο το object, για να αναφερθούμε σε ένα υπάρχον object αντι να το αντιγράφουμε. π.χ. Το **ref(o3)** θα επιστρέψει ενα reference στο object o3.

Functions των objects της γλώσσας

Η γλώσσα πρέπει να υποστηρίζει την δήλωση συναρτήσεων ως πεδία στο σώμα των objects. Οι συναρτήσεις **πρέπει** να επιστρέφουν ένα value και δεν δέχονται παραμέτρους. Σε περίπτωση που δεν θέλετε επιστρεφόμενη τιμή επιστρέφετε το keyword **none**.

Η δημιουργία συναρτήσεων γίνεται με δύο τρόπους. Με τα keywords **lambda** και **func**.

- Το **func** keyword χρησιμοποιείται για την δήλωση μιας συνάρτησης ακολουθούμενο απο το id της και το σώμα της και αποτελεί μέλος ενος object.

  ```
  func(id){body}
  ```

- Το **lambda** keyword ακολουθείται αμέσως με το σώμα της συνάρτησης και πρέπει να το αποθηκεύσουμε σε ένα key για να είναι μέλος ενός object.

  ```
  lambda {body}
  ```

Το συντακτικό **key(id) = lambda{body}** είναι ισοδύναμο με το **func(id){body}**.

<u>Αναφορά σε πεδία ενός object μέσα απο ένα function.</u>

Μέσα στο σώμα ενός function μπορούμε να αναφερθούμε στα μέλη ενός object με τα keywords **self** και **arg**. Προσοχή στο γεγονός οτι το function θα εκτελείται απο τον receiver.

- Το **self** keyword επιστρέφει την μεταβλητή με το id που βρίσκεται στις παρενθέσεις η οποία βρίσκεται στο **receiving** object που θα τρέξει την συνάρτηση.

  ```
  self(x) // επιστρέφει το μέλος με id = "x" που βρίσκεται στο receiving object
  ```

- Το **arg** keyword επιστρέφει την μεταβλητή με το id που βρίσκεται στις παρενθέσεις η οποία βρίσκεται στο **sender** object που θα στείλει το μήνυμα .

  ```
  arg(y) // επιστρέφει το μέλος με id = "y" που βρίσκεται στο message object
  ```

Μεταφορά μηνυμάτων

Η μεταφορά ενός μηνύματος απο ένα object προς ένα receiving object πρέπει να γίνεται με τον ακόλουθο τρόπο, χρησιμοποιώντας τον operator **<<**:

```
rec_object << msg_object;  // το msg_object στέλνει μήνυμα στο rec_object
```

- Το *rec_object* είναι το όνομα του object που θα λάβει το μήνυμα και το *msg_object* θα είναι το object που θα δίνει τα ορίσματα του στο *rec_object* για να εκτελέσει το μήνυμα.
- Για τη δημιουργία ενός receiving object ακολουθούμε το ίδιο συντακτικό με τα objects που είδαμε παραπάνω, πρέπει όμως να υπάρχει, ως πεδίο του object, μια function η οποία θα εξυπηρετήσει το αίτημα του μηνύματος.
- Το *msg_object* θα πρέπει να έχει μια επιπλέον παράμετρο `call(id).` Πρέπει να ορίσετε το keyword `call` έτσι ώστε όταν γίνει το message passing να κληθεί στον receiver η συνάρτηση με id εντός των παρενθέσεων.

Παρακάτω βλέπουμε την δήλωση ενός receiving object του οποίου η συνάρτηση "printf" ,όταν κληθεί θα τυπώνει τις παραμέτρους που περνάμε. Το **nl** keyword αναπαριστά το new line character.

```
let printf_impl = object [            // δημιουργούμε το object που θα λάβει ένα message
       func("printf") {               // η συνάρτηση που πρέπει να γίνει called
            for (auto& v : args_list) // iterate την λίστα με τα arguments στο message
               std::cout << v << nl;  // print τα argument μαζί με ένα new line στο τέλος
            return none;
       }
   ];
```

Προσθέτωντας το πεδίο `call("printf")` στο παρακάτω message object το μήνυμα είναι έτοιμο προς αποστολή στο **`printf_impl`** object.

```
o3 = object [ call("printf"), values 1, "2", true, -3.14 ];
printf_impl << o3; // Καλείται η printf στον receiver και παιρνει ένα ένα τα ορίσματα του
o3 εκτυπώνωντας 1,"2",true,-3.14
```

Εκτέλεση συναρτήσεων των μηνυμάτων

Για να προσθέτε τη δυνατότητα στα receiving objects να εκτελούν τις συναρτήσεις που στέλνουμε ως μηνύματα και να διατρέχουν εύκολα τις παραμέτρους του message, θα πρέπει να ορίσετε τα παρακάτω keywords:

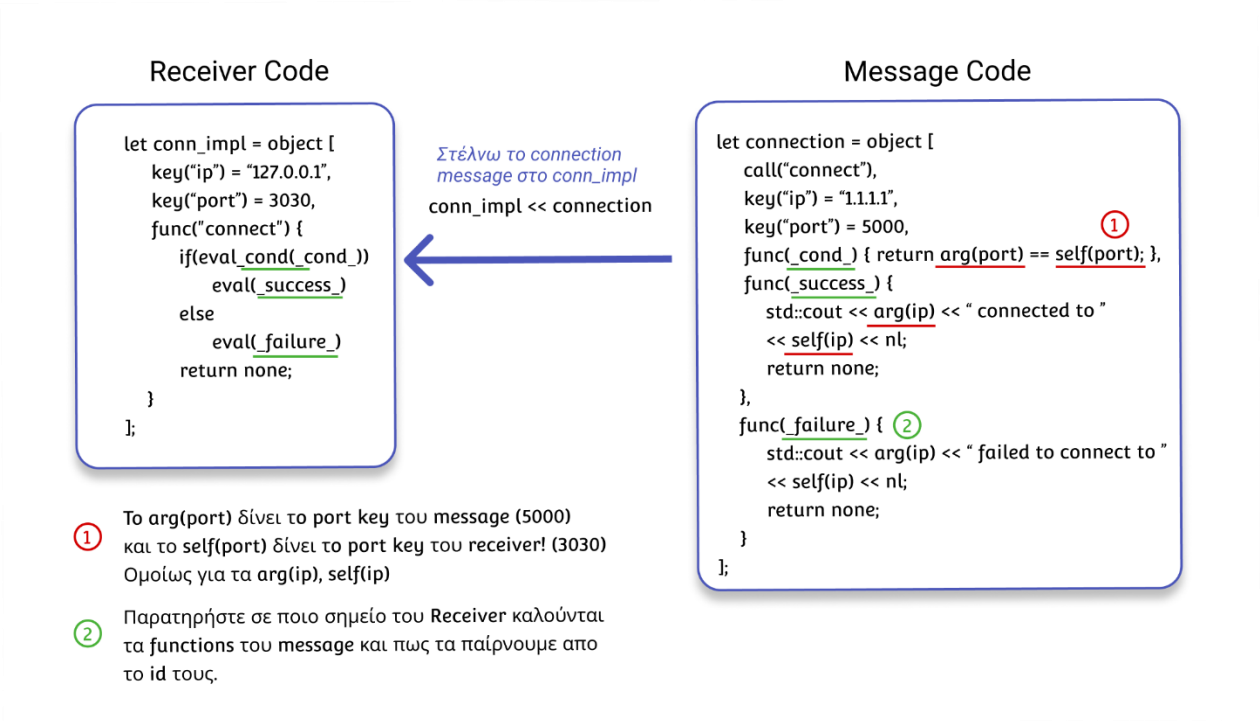
- `args_list`: Επιστρέφει μια iteratable λίστα με όλα τα arguments που λαμβάνει ο receiver.
- `eval`: Παίρνει ως όρισμα το id μιας συνάρτησης που περάσαμε ως argument και την εκτελεί. π.χ. :

  ```
  eval("mid"); // Εκτελεί (αν υπάρχει) την συνάρτηση με id "mid" απο το sending object.
  ```

- `eval_cond`: Παίρνει ως όρισμα το id μιας συνάρτησης που περάσαμε ως λογική συνθήκη επιστρέφοντας την Boolean αποτίμηση της συνάρτησης (Ακολουθεί παράδειγμα χρήσης των keywords.)

Σύνθετο παράδειγμα χρήσης της γλώσσας

Στο παρακάτω παράδειγμα έχουμε ορίσει ένα message και ένα receiving object που χρησιμοποιούν τα στοιχεία της γλώσσας.



Το connection object καλεί την συνάρτηση “connect” του receiver περνώντας όλα τα πεδία του στο μήνυμα. Ο receiver έπειτα εκτελεί τις συναρτήσεις του connection object βασει το id τους ανάλογα με το αποτέλεσμα της συνθήκης που επιστρέφει η συνάρτηση *_cond_*. Εν τέλει αποτιμείται η *_failure_* εφόσον τα πεδία “port” των object που συγκρίνονται είναι διαφορετικά.

Hints

Για να μετατρέψετε το συντακτικό της γλώσσας σε valid C++ χρησιμοποιήστε:

- Τη δυνατότητα για operator overloading που σας προσφέρει η C++, δίνοντας μεγάλη προσοχή στην προτεραιότητα των operators.
  - `operator[]`
    Για τον ορισμό object
  - `operator,`
    Για να μαζεύετε εκφράσεις που έχουν κόμμα ανάμεσα τους

- Δημιουργία προσωρινών στιγμιότυπων ως επιστρεφόμενα αποτελέσματα, αλλά και ως βοηθητικά στιγμιότυπα σε εκφράσεις.

- Τα template containers που προσφέρει η STL για να αποθηκεύετε τα διαφορετικά αντικείμενα του προγράμματος.

- Αρκετά τον preprocessor αφού λέξεις κλειδιά της γλώσσας όπως **object**, **key**, ***func*** *κτλ.* θα είναι macros που θα κρύβουν μετατροπές σε strings, κλήσεις συναρτήσεων, κάποιους operators ή και βοηθητικά προσωρινά στιγμιότυπα.