Παρουσίαση της γλώσσας προγραμματισμού Rust

Ιούνιος 2017

Η παρούσα εργασία αποτελεί προϊόν αποκλειστικά δικής μου προσπάθειας. Όλες οι πηγές που χρησιμοποιήθηκαν περιλαμβάνονται στη βιβλιογραφία και γίνεται ρητή αναφορά σε αυτές μέσα στο κείμενο όπου έχουν χρησιμοποιηθεί.

Περιεχόμενα

Κατάλογος Σχημάτων

Περίληψη

Παρακάτω υπάρχει μία γενική παρουσίαση της Rust, μιας σύγχρονης γλώσσας προγραμματισμού. Δεν απαιτούνται υψηλές γνώσεις προγραμματισμού από τους αναγνώστες για να γίνει κατανοητό το κείμενο. Μετά από μία αναδρομή στην ιστορία των γλωσσών προγραμματισμού και τα κύρια χαρακτηριστικά τους, γίνεται αναφορά στους λόγους ή προβλήματα που οδήγησαν στην κατασκευή της Rust, ενώ στο τελευταίο κεφάλαιο υπάρχει μία προσπάθεια πρόβλεψης της μελλοντικής επιτυχίας της. Σκοπός της εργασίας είναι να δώσει μία εναρκτήρια βάση σε όσα άτομα επιθυμούν να γνωρίσουν τη συγκεκριμένη γλώσσα (ή ακόμα και άλλες) ώστε στη συνέχεια να μπορέσουν να αναζητήσουν περαιτέρω πληροφορίες από μόνοι τους.

1 Γλώσσες προγραμματισμού

1.1 Ιστορία των γλωσσών προγραμματισμού

Η αρχή των σύγχρονων γλωσσών προγραμματισμού είναι συνυφασμένη με τους πρώτους ηλεκτρονικούς υπολογιστές, που δημιουργήθηκαν τη δεκαετία του 1940.[1] Για να λειτουργήσουν τα πρώιμα συστήματα, έπρεπε ο προγραμματισμός τους να γίνει σε γλώσσες χαμηλού επιπέδου (assembly), που ήταν κοντά στη γλώσσα μηχανής του υπολογιστή. Ωστόσο η χρήση της assembly ήταν χρονοβόρα, επιρρεπής σε λάθη και μη φορητή, με αποτέλεσμα να είναι απαραίτητο να ξαναγραφτεί ένα λογισμικό για να μεταφερθεί σε κάποιο άλλο σύστημα.

Για να αντιμετωπιστούν αυτά τα προβλήματα δημιουργήθηκαν σταδιακά οι γλώσσες υψηλού επιπέδου, οι οποίες είναι ανεξάρτητες από το hardware στο οποίο τρέχουν, ενώ παρέχουν τρόπους γραφής αρκετά πιο κοντά στον ανθρώπινο τρόπο σκέψης. Οι πρώτες από αυτές ήταν πειραματικές ή ακαδημαϊκού ενδιαφέροντος και δεν έτυχαν ευρείας εφαρμογής. Το 1957 δημιουργήθηκε η FORTRAN από τον John Backus της IBM και πολύ γρήγορα καθιερώθηκε, καθώς επέτρεπε τη δημιουργία προγραμμάτων με πολύ λιγότερες γραμμές κώδικα σε σχέση με τις τότε μεθοδολογίες.

Μέσα στα επόμενα χρόνια εμφανίστηκαν πολλές γλώσσες που σταδιακά πρόσθεσαν τους υπόλοιπους λίθους του τωρινού προγραμματισμού:

- **Lisp** (1958). Εισήγαγε για πρώτα φορά δυνατότητες μεταπρογραμματισμού μέσω των συμβολικών εκφράσεων (S-expressions), επιτρέποντας τη δημιουργία ισχυρών προγραμμάτων που μπορούν να επεξεργαστούν προγράμματα σαν δεδομένα. Ήταν η πρώτη γλώσσα που περιείχε έναν garbage collector, με σκοπό την αυτόματη ελευθέρωση της μνήμης που χρησιμοποιήθηκε από

[1] Πιο πριν υπήρχαν θεωρητικά μαθηματικά μοντέλα και μη ηλεκτρονικές υπολογιστικές συσκευές

το πρόγραμμα χωρίς να χρειάζεται να οριστεί χειροκίνητα. Η κύρια εφαρμογή της είναι στην Τεχνητή Νοημοσύνη.

- **ALGOL** (1958-1960). Εισήγαγε αρκετές καινοτομίες όπως τα μπλοκ εντολών (begin ... end), τις εμφωλευμένες συναρτήσεις, τη λεκτική εμβελειοθέτηση (lexical scope), δηλαδή τη δυνατότητα το όνομα μιας μεταβλητής να είναι ορισμένο μόνο σε ένα σημείο του προγράμματος. Επίσης βοήθησε στη μέθοδο περιγραφής των γλωσσών προγραμματισμού με τη καθιέρωση της μορφής Backus-Naur.

- **COBOL** (1959). Δημιουργήθηκε για εμπορικές χρήσεις. Το συντακτικό της ήταν σχεδιασμένο να θυμίζει εκφράσεις της αγγλικής γλώσσας, προκειμένου να είναι πιο εύκολη η χρήση της.

- **APL** (1964). Παρείχε ένα πολύ περιεκτικό συντακτικό, με σκοπό ο χρήστης να μην πληκτρολογεί πολλούς χαρακτήρες.

- **BASIC** (1964). Σκοπός της ήταν να κάνει προσιτό τον προγραμματισμό σε άτομα που δεν είχαν επιστημονικές ή μαθηματικές γλώσσες, όπως συνηθιζόταν πιο πριν. Ήταν ιδιαίτερα δημοφιλής, με πάρα πολλές διαλέκτους και διαφορετικές υλοποιήσεις τις επόμενες δεκαετίες.

- **Simula** (1965). Η πρώτη αντικειμενοστραφής γλώσσα. Εισήγαγε αντικείμενα, κλάσεις, κληρονομικότητα, απορρέουσες κλάσεις, εικονικές συναρτήσεις, coroutines (χρήσιμες για πολυνηματικό προγραμματισμό) καθώς και πολυμορφισμό υποτύπων (subtyping), μια νέα μορφή προγραμματισμού. Όλες οι μετέπειτα αντικειμενοστραφείς γλώσσες είναι άτυποι απόγονοι της.

  Από τη δεκαετία του 1970 και έπειτα έκαναν την εμφάνιση τους αρκετές γλώσσες που χρησιμοποιούνται κατά κόρον ακόμα και σήμερα. Σε ακαδημαϊκό επίπεδο έφεραν αρκετές καινοτομίες, αν και όχι τόσο συνταρακτικές όσο οι γλώσσες της προηγούμενης κατηγορίας· το κυριότερο πλεονέκτημα τους ήταν η πρακτικότητα και η ευκολία χρήσης τους από ένα πιο ευρύ κοινό, εν μέρει χάρη στην εξέλιξη της υπολογιστικής τεχνολογίας.

- **Pascal** (1970). Είχε εκπαιδευτικό χαρακτήρα, με σκοπό να διδάξει τα πλεονεκτήματα του δομημένου προγραμματισμού.

- **C** (1972). Δημιουργήθηκε στα Bell Labs για την υλοποίηση του λειτουργικού Unix. Είχε εξαιρετική διάδοση και είναι ακόμα και σήμερα από τις πιο δημοφιλείς γλώσσες προγραμματισμού, με πάρα πολλά λογισμικά να είναι γραμμένα σε αυτή. Αρκετές γλώσσες προγραμματισμού βασίζονται σε αυτή και τα συντακτικά της χαρακτηριστικά. Υποστηρίζεται από πληθώρα μεταγλωττιστών καθώς και όλα τα λειτουργικά ή αρχιτεκτονικές υπολογιστών.

- **Smalltalk** (1972).

- **Prolog** (1972). Γλώσσα λογικού προγραμματισμού, προορισμένη για χρήση στον τομέα της Τεχνητής Νοημοσύνης.

- **SQL** (1974). Είναι βασισμένη σε σχεσιακά μοντέλα και χρησιμοποιείται στις περισσότερες κοινές βάσεις δεδομένων. Υπάρχουν πολλές διαφορετικές υλοποιήσεις, συνήθως ασύμβατες μεταξύ τους.

- **C++** (1983). Αρχικά επέκτεινε την C με χαρακτηριστικά του αντικειμενοστραφή προγραμματισμού. Πολύ σύντομα απέκτησε μεγάλη δημοτικότητα καθώς προσέφερε περισσότερες δυνατότητες από τον πρόγονο της για την δημιουργία μεγάλων και πολύπλοκων προγραμμάτων. Είναι από τις πιο διαδεδομένες αντικειμενοστραφείς γλώσσες και συνεχίζει να αναπτύσσεται ενεργά.

- **Ada** (1983)

  Η δεκαετία του 1990 σημαδεύτηκε από την ραγδαία άνοδο του Διαδικτύου. Πολλές γλώσσες εκείνης της εποχής δημιουργήθηκαν για να προσφέρουν τα απαραίτητα εργαλεία για την εξέλιξη του. Ταυτόχρονα η ανάπτυξη των γλωσσών απέκτησε το πλεονέκτημα της εύκολης συνεργασίας μέσω του Διαδικτύου, σε αντίθεση με την απομονωμένη ανάπτυξη από διαφορετικές ομάδες που υπήρχε πιο πριν.

- **Python** (1991)
- **Java** (1995)
- **JavaScript** (1995)
- **D** (1999)

Η μελλοντική πορεία των γλωσσών προγραμματισμού (κρίνοντας από την τελευταία δεκαετία) φαίνεται να κινείται προς αποδοτικές γλώσσες που προσφέρουν μεγαλύτερο βαθμό ευχρηστίας (Go, Rust, Julia, Swift), γλώσσες που μπορούν να κάνουν compile (και) σε JavaScript, προκειμένου να λειτουργούν μέσα σε περιηγητές διαδικτύου ή περιβάλλοντα που παρέχουν παρόμοιες δυνατότητες (CoffeeScript, TypeScript, Kotlin κ.ά.)

1.2 Χαρακτηριστικά γλωσσών προγραμματισμού

1.2.1 Τρόπος εκτέλεσης

Η FORTRAN ήταν η πρώτη σημαντική γλώσσα που χρησιμοποίησε μεταγλωττιστή (compiler) για την εκτέλεση της· πιο πριν είχαν υπάρξει διερμηνευμένες γλώσσες (interpreted languages). Η κύρια διαφορά μεταξύ διερμηνευτών και μεταγλωττιστών είναι πως οι πρώτοι εκτελούν το πρόγραμμα απευθείας όταν γίνεται η εκτέλεση του, ενώ οι τελευταίοι μετατρέπουν πρώτα τον κώδικα σε κάποια εκτελέσιμη μορφή η οποία στη συνέχεια θα ”τρέξει” στους υπολογιστές. Η ιδέα των διερμηνευτών είναι πιο απλή, ωστόσο οι μεταγλωττιστές συνήθως παράγουν ταχύτερο και πιο αποδοτικό κώδικα, χρησιμοποιώντας διάφορες βελτιστοποιήσεις κατά τη διάρκεια της μεταγλώττισης.

1.2.2 Υποδείγματα προγραμματισμού

Υπάρχουν αρκετοί τρόποι προγραμματισμού, ο καθένας με τα δικά του πλεονεκτήματα ή μειονεκτήματα. Η χρήση τους διαφέρει ανάλογα με τους τομείς, τα περιβάλλοντα εργασίας ή τον προορισμό των προγραμμάτων. Ακολουθεί μία γρήγορη παρουσίαση των τεσσάρων πιο δημοφιλών:

- **Προστακτικός** *(imperative)* προγραμματισμός

  Ο προστακτικός ή διαδικαστικός είναι ένας από τους παλιότερους και πιο συνηθισμένους τρόπους προγραμματισμού. Στον προστακτικό προγραμματισμό η έμφαση δίνεται στον τρόπο εκτέλεσης των υπολογισμών, δηλαδή στο *“πώς;”* (*πώς θα επιλυθεί το πρόβλημα;*). [1] Ο συγκεκριμένος τρόπος προγραμματισμού μπορεί να εκφραστεί από την εξίσωση του Wirth:

  Προγράμματα = Αλγόριθμοι + Δομές Δεδομένων

- **Αντικειμενοστραφής** *(object-oriented)* προγραμματισμός

  Ο αντικειμενοστραφής προγραμματισμός έχει ως βάση του την αρχή της κλάσης (class). Κάθε κλάση περιλαμβάνει πολλούς ορισμούς ή διαδικασίες που είναι ενιαία και συμπεριφέρονται σαν μια μονάδα. Τα αντικείμενα (objects) που περιέχονται σε μία κλάση είναι ανεξάρτητα από αυτήν ή τη διάρκεια ζωής της και μπορούν να δημιουργηθούν έξω από το μπλοκ στο

οποίο ορίζονται. [2]

Ο αντικειμενοστραφής προγραμματισμός προσφέρει το χαρακτηριστικό της αφαίρεσης δεδομένων (data abstraction). Αυτό επιτρέπει την επαναχρησιμοποίηση κώδικα μέσα στα προγράμματα, αλλά και τον ευκολότερο επανασχεδιασμό της δομής τους σε περίπτωση που είναι απαραίτητο. Χρησιμοποιείται συχνά σε πολύπλοκα προγράμματα.

- **Συναρτησιακός** *(functional)* προγραμματισμός

  Ο συναρτησιακός προγραμματισμός είναι υπό μία έννοια ένα αρκετά πιο απλό και "καθαρό" υπόδειγμα από τον προστακτικό προγραμματισμό. Αυτό οφείλεται στις μαθηματικές του ρίζες, όπως η θεωρία των συναρτήσεων. [3]

- **Λογικός** *(logic)* προγραμματισμός

  Ο λογικός προγραμματισμός είναι αρκετά διαφορετικός από τα τρία προηγούμενα υποδείγματα. Είναι πολύ χρήσιμος στις περιπτώσεις που έχουμε προβλήματα που απαιτούν εξαγωγή γνώσης από βασικά γεγονότα ή συσχετίσεις. Έχει συγκεκριμένες δυνατότητες και πιθανόν να μην ανταποκρίνεται καλά σε γενικότερες περιοχές υπολογισμών. [3]. Μπορεί να εκφραστεί από την εξίσωση του Kowalski:

  Αλγόριθμος = Λογική + Έλεγχος

Πολλές γλώσσες προγραμματισμού δεν είναι περιορισμένες σε ένα μόνο υπόδειγμα και υποστηρίζουν δύο ή περισσότερα.

1.2.3 Σύστημα τύπων

- Στατικοί και δυναμικοί τύποι

  Στις γλώσσες με στατικούς τύπους *(static typing)* ο ελέγχος των τύπων γίνεται κατά τη διάρκεια της μεταγλώττισης. Αντίθετα, στις γλώσσες με δυναμικούς τύπους *(dynamic typing)* ο έλεγχος γίνεται κατά τη διάρκεια της εκτέλεσης του προγράμματος.

- Ισχυροί και ασθενείς τύποι

- Οι καινούργιες γλώσσες συνήθως προσπαθούν να ακολουθούν κάποια από τα συντακτικά χαρακτηριστικά των προηγούμενων όπου είναι δυνατόν, λόγω εξοικείωσης των προγραμματιστών με αυτά. Ένα απλό παράδειγμα είναι τα άγκιστρα { } τα οποία προήλθαν από τη C, υποδηλώνουν την αρχή και το τέλος ενός block εντολών και έχουν υιοθετηθεί από τις περισσότερες γλώσσες που εμφανίστηκαν έπειτα από αυτή.

- Οι περισσότερες γλώσσες προγραμματισμού πλέον είναι διαθέσιμες υπό permissive[2] άδειες ανοιχτού κώδικα, που επιτρέπουν πρακτικά οποιαδήποτε αλλαγή ή προσθήκη στον κώδικα. Αυτό γίνεται προκειμένου οι γλώσσες να διαδοθούν και να χρησιμοποιηθούν όσο γίνεται περισσότερο, αφού τυχόν περιορισμοί θα αποτρέψουν τους επίδοξους προγραμματιστές.

[2] Υπάρχουν δύο “στρατόπεδα” στις άδειες ανοιχτού κώδικα: οι copyleft άδειες, που απαιτούν την αναδιανομή του κώδικα σε περίπτωση αλλαγής του και οι permissive άδειες που δεν θέτουν ιδιαίτερους περιορισμούς στη χρήση του διαθέσιμου κώδικα.

2 Εισαγωγή στη Rust

2.1 Ιστορία της Rust

Η Rust ξεκίνησε το 2006, από τον Graydon Hoare που εργαζόταν στην Mozilla. Αρχικά, ήταν απλά ένα προσωπικό "χόμπυ" και δεν ήταν σαφής ο σκοπός που θα επιτελούσε. Το 2009 η Mozilla άρχισε να προσφέρει χρηματική βοήθεια και ανέθεσε κάποιους εργαζόμενους πλήρους απασχόλησης με σκοπό να δουλέψουν στη γλώσσα. Η επίσημη ανακοίνωση έγινε το 2010. [4] Επειδή το κύριο "προϊόν" της Mozilla είναι ο Firefox, ο γνωστός περιηγητής διαδικτύου, και τα υπεύθυνα άτομα είχαν εμπειρία με τα μειονεκτήματα και τις ιδιοτροπίες της C++ στην κατασκευή λογισμικών μεγάλου μεγέθους, σιγά-σιγά η γλώσσα άρχισε να κινείται προς την κατεύθυνση του προγραμματισμού συστημάτων (system programming) και να γίνεται ένας άτυπος ανταγωνιστής της C και της C++.

Σχήμα 1: Το λογότυπο της Rust[3]

Ο πρώτος compiler της Rust ήταν γραμμένος σε άλλη γλώσσα, την Ocaml. Κάποια στιγμή ξεκίνησε να αναπτύσσεται ένας compiler γραμμένος σε Rust, ο rustc, χρησιμοποιώντας το LLVM σαν βάση πάνω στην οποία στηρίχτηκε. Η πρώτη έκδοση του rustc που ήταν έτοιμη για χρήση (δηλαδή κατάφερε να κάνει compile τον εαυτό του, bootstrapping) εμφανίστηκε το 2011. Τα μετέπειτα χρόνια χρησίμευσε πολύ στους δημιουργούς της Rust και σαν πεδίο δοκιμών, καθώς μέσω της ανάπτυξης ενός σύνθετου λογισμικού μπορούσαν να καταλάβουν τα μειονεκτήματα της γλώσσας και να τα αντιμετωπίσουν κατάλληλα.

Η Rust 0.1 κυκλοφόρησε τον Ιανουάριο του 2012. Τα επόμενα δυόμιση χρόνια πέρασε από μεγάλες αλλαγές και πειραματισμούς με πολλά κύρια ή συντακτικά χαρακτηριστικά να αλλάζουν ριζικά σε εκείνη την περίοδο. Η Rust 1.0 κυκλοφόρησε τον Μάϊο του 2015 και εγγυήθηκε σταθερότητα και συμβατότητα των επόμενων εκδόσεων με τη συγκεκριμένη έκδοση (ώστε να μην χρειάζονται μεγάλες αλλαγές σε πορεία χρόνου για να συνεχίσουν να δουλεύουν τα προγράμματα). Σε αντίθεση με άλλες γλώσσες προγραμματισμού συστημάτων, όπως η C++ ή η C, στις οποίες οι κυκλοφορίες νέων εκδόσεων απέχουν χρονικά πολύ η μία από την άλλη, η Rust κυκλοφορεί νέες εκδόσεις κάθε 6 εβδομάδες.

Σήμερα το μέλλον της Rust φαντάζει αρκετά ελπιδοφόρο, με την χρήση της να αυξάνεται κατακόρυφα και ολοένα και περισσότερες μεγάλες εταιρίες να την υιοθετούν (περισσότερα στοιχεία στο κεφάλαιο του οικοσυστήματος). Μία ανάλυση για την πιθανή πορεία της μπορεί να βρεθεί στο τελευταίο κεφάλαιο.

[3] Διαθέσιμο υπό άδεια Creative Commons Attribution *(CC-BY)*

2.2 Προβλήματα με τις προηγούμενες γλώσσες προγραμματισμού

Η Rust σχεδιάστηκε με σκοπό να βοηθήσει στην αποφυγή λαθών τα οποία εμφανίζονται συχνά στις ήδη διαδεδομένες γλώσσες προγραμματισμού. Παρότι σχεδόν όλα τα λογισμικά που χρησιμοποιούνται σε κρίσιμα συστήματα (για βιομηχανική, ιατρική ή προσωπική χρήση) και απαιτούν γρήγορη απόκριση είναι γραμμένα σε C ή C++, αυτές δεν παρέχουν την απαραίτητη ασφάλεια χρήσης, καθώς είναι επιρρεπείς από αρκετές κατηγορίες σφαλμάτων.

2.2.1 Διαχείριση μνήμης

- **Απρόσμενη συμπεριφορά**

```
1 int main(int argc, char **argv) {
2     unsigned long a[1];
3     a[3] = 0x7ffff7b36cebUL;
4     return 0;
5 }
```

Το παραπάνω μικρό πρόγραμμα παρότι μεταγλωττίζεται παρουσιάζει απρόσμενη συμπεριφορά *(undefined behavior)*,[4] καθώς επιστρέφεται το 3ο στοιχείο του πίνακα, αλλά σε αυτό έχουμε ορίσει μία καινούργια θέση μνήμης. Έτσι το πρόγραμμα επιστρέφει στοιχεία που ανήκουν σε κάποιο άλλο εκτελούμενο πρόγραμμα, θέτοντας σε κίνδυνο την ασφάλεια του συστήματος. [5]

Τα μειονεκτήματα αυτά, παρότι γνωστά, συνεχίζουν να απασχολούν τους προγραμματιστές, καθώς δεν είναι καθόλου εύκολο να αντιμετωπιστούν αποτελεσματικά. Πολλά γνωστά και καλά ελεγμένα λογισμικά είχαν (και συνεχίζουν να έχουν) bugs εξαιτίας αυτών. [6]

2.2.2 Concurrency

Την εποχή που δημιουργήθηκαν αρκετές δημοφιλείς γλώσσες δεν υπήρχαν πολυπύρηνα συστήματα, ούτε γινόταν χρήση πολυνηματισμού (multithreading). Γι' αυτό το λόγο, οι περισσότερες γλώσσες δεν είναι φτιαγμένες για να εκμεταλλευτούν τους πόρους των σύγχρονων επεξεργαστών. Έτσι είναι ευάλωτες σε συνθήκες ανταγωνισμού (data races).

Τα data races συμβαίνουν όταν δύο εντολές από διαφορετικά νήματα προσπαθούν να προσπελάσουν την ίδια θέση μνήμης, τουλάχιστον μία από αυτές είναι

[4] Σε κάποια συστήματα παρουσιάζεται segmentation fault και το πρόγραμμα σταματάει να εκτελείται.

εντολή εγγραφής και δεν υπάρχει συγχρονισμός που να καθορίζει μία συγκεκριμένη σειρά ανάμεσα σε αυτές τις προσπελάσεις. [7] Αυτό δυσκολεύει κατά πολύ την δημιουργία λογισμικού που κάνει χρήση πολλών νημάτων.

3 Κύρια χαρακτηριστικά της Rust

Πριν προχωρήσουμε στα κύρια χαρακτηριστικά της Rust, καλό είναι να εξηγηθούν μερικοί από τους προγραμματιστικούς όρους που χρησιμοποιούνται στη συνέχεια:

Scoping (εμβελειοθέτηση): Εμβέλεια ονομάζεται ο τρόπος με τον οποίο ρυθμίζεται η δυνατότητα οντοτήτων, όπως μεταβλητές, ετικέτες, τύποι ή διαδικασίες, στις οποίες έχει δοθεί ένα όνομα, να επιδρούν σε ένα πρόγραμμα. [2] Για παράδειγμα, μία μεταβλητή i που δηλώνεται στην main θα είναι εκτός εμβέλειας σε μία συνάρτηση test_scope που έχει επίσης μία μεταβλητή i (πιθανώς με διαφορετική τιμή).

Υπάρχουν δύο κύρια είδη εμβελειοθέτησης, η στατική ή λεκτική (static ή lexical scoping) και η δυναμική (dynamic scoping). Η πρώτη είναι η πιο συνηθισμένη, ενώ η δεύτερη χρησιμοποιείται κυρίως στη Lisp.

3.1 Ιδιοκτησία (Ownership)

Ίσως το κυριότερο χαρακτηριστικό της Rust είναι αυτό της ιδιοκτησίας. Με απλά λόγια, σημαίνει πως κάθε δέσμευση μνήμης έχει έναν μόνο ιδιοκτήτη και αυτός ο ιδιοκτήτης πρέπει να την αποδεσμεύσει όταν δεν χρειάζεται πλέον. Σε περίπτωση που υπάρχουν περισσότερες από μία συναρτήσεις που ανταλλάσσουν δεδομένα ή μεταβλητές, η συνάρτηση που δέχεται τα δεδομένα γίνεται ο νέος ιδιοκτήτης και ο παλιός αφαιρείται. Επομένως η νέα συνάρτηση είναι τώρα υπεύθυνη για την καταστροφή των πόρων. Σε περίπτωση που ο ιδιοκτήτης ξεφύγει από την εμβέλεια (scope) του προγράμματος, η τιμή καταστρέφεται. [8]

Τα πλεονεκτήματα αυτού του μοντέλου βοηθούν τόσο και σε ασφάλεια μνήμης όσο και σε πολυνημάτωση. Τα περισσότερα από τα σφάλματα που αναφέρθηκαν νωρίτερα δεν υφίστανται πλέον, καθώς πέρα από τον ιδιοκτήτη δεν μπορούν άλλοι χρήστες να προσπελάσουν, αλλάξουν ή διαγράψουν κάποια τιμή. Έτσι αποφεύγεται τυχόν απρόσμενη συμπεριφορά. Επίσης το μοντέλο αυτό είναι thread-safe, οπότε δεν πάσχει από data races, που δυσκολεύουν τον προγραμματισμό στην C++.

3.2 Δανεισμός (Borrowing)

Χάρη στον “δανεισμό” μπορούν να υπάρχουν πολλαπλές αναφορές (references) σε κάποιον πόρο χωρίς να παραβιάζεται η αρχή της ιδιοκτησίας. Οι αναφορές είναι παρόμοιες με τους δείκτες (pointers) στην C.

Οι κυριότεροι κανόνες του συγκεκριμένου χαρακτηριστικού: [9]

- Ένας δανεισμός δεν μπορεί να διαρκεί περισσότερο από το scope του ιδιοκτήτη του.

- Ένας πόρος μπορεί να έχει μόνο ένα είδος δανεισμού μία συγκεκριμένη στιγμή. Τα είδη αυτά είναι:

  - Πολλαπλοί δανεισμοί για ανάγνωση (Multiple Reader Borrows): πολλές αναφορές (&T) σε έναν πόρο.

  - Μοναδικός δανεισμός για εγγραφή (Single Write Borrow): μόνο μία μεταβλητή αναφορά σε έναν πόρο.

Έτσι αποφεύγεται το πρόβλημα των data races που υπάρχει σε άλλες γλώσσες (αφού σε περίπτωση που συμβαίνει εγγραφή σε κάποιον πόρο δεν μπορεί να εκτελείται κάποια άλλη διαδικασία ταυτόχρονα).

3.3 Χρόνος ζωής (Lifetimes)

Κάθε πόρος στη Rust έχει έναν συγκεκριμένο χρόνο ζωής. Αυτό ισχύει και για τις αναφορές και για τους δανεισμούς, προκειμένου να αποφευχθεί το πρόβλημα των αιωρουμένων δεικτών (dangling pointers) που μπορεί να συνεχίσουν να λειτουργούν και να δείχνουν σε μία θέση μνήμης που πλέον δεν υπάρχει. [9]

Ένα ελαφρώς πολύπλοκο θέμα είναι πως ο χρόνος ζωής μιας αναφοράς μπορεί να είναι διαφορετικός από τον χρόνο ζωής του δανεισμού που τις αντιπροσωπεύει. Αυτό συμβαίνει γιατί μπορεί παραπάνω από μία αναφορά να βασίζεται σε έναν συγκεκριμένο δανεισμό, οπότε ο τελευταίος θα έχει μεγαλύτερο χρόνο ζωής. [10]

4 Άλλα πλεονεκτήματα της Rust

4.1 Απόδοση

Η Rust, αφού στοχεύει τον τομέα του προγραμματισμού συστημάτων, έχει λάβει υπόψη της την απόδοση και ταχύτητα του παραγόμενου κώδικα.

Πρέπει να αναφερθεί ότι είναι δύσκολη μία απολύτως αξιόπιστη σύγκριση της ταχύτητας μεταξύ δύο γλωσσών προγραμματισμού. Το να ξαναγραφτεί κώδικας από μία γλώσσα σε μία άλλη δεν είναι εύκολο, καθώς δεν υπάρχει απαραίτητα ένας κοινός ιδανικός τρόπος υλοποίησης. Για παράδειγμα, είναι πιθανό κάποιος να προσπαθήσει να ξαναγράψει σε Rust ένα πρόγραμμα που ήταν αρχικά γραμμένο σε C++, αλλά από άγνοια να μην χρησιμοποιήσει τη βέλτιστη δυνατή λύση. Παρομοίως, το ίδιο μπορεί να συμβεί και στην αντίθετη περίπτωση, από Rust σε C++. Επίσης οι περισσότερες συγκρίσεις γίνονται μεταξύ μικρών και σχετικά απλοϊκών κομματιών κώδικα, που μπορεί να διαφέρουν από τη χρήση σε μία εφαρμογή μεγαλύτερου μεγέθους ή πολυπλοκότητας. [11]

Παρακάτω παρουσιάζονται κάποια ενδεικτικά benchmarks, λάβετε υπόψη ότι μπορεί να πάσχουν από τα προβλήματα που αναφέρθηκαν παραπάνω:

- The Computer Language Benchmarks Game: *https://benchmarksgame.alioth.debian.org/*
- kostya/benchmarks: *https://github.com/kostya/benchmarks*

Τα παραπάνω benchmarks περιέχουν διάφορα μικρά προγράμματα υλοποιημένα σε διάφορες γλώσσες και μία σύγκριση της ταχύτητας και χρήσης μνήμης μεταξύ αυτών. Παρά την απλοϊκή σύγκριση, είναι φανερό πως η Rust βρίσκεται συνήθως στις 5 πρώτες γλώσσες με βάση την ταχύτητα.

4.2 Εύκολη αλληλεπίδραση με άλλες γλώσσες

Η Rust έχει λάβει υπόψην της πως τεράστια ποσότητα κώδικα θα συνεχίσει να είναι γραμμένος σε άλλες γλώσσες και γι’ αυτό έχει κάνει προσπάθειες για εύκολη αλληλεπίδραση με αυτές. Με το FFI *(foreign function interface)* της, η Rust μπορεί να κληθεί από άλλες γλώσσες όπως η C. Έτσι μπορεί κάποιο πρόγραμμα που είναι γραμμένο σε αυτές να αρχίσει να χρησιμοποιήσει σταδιακά τη Rust σε κάποια κομμάτια κώδικα, χωρίς να απαιτείται να ξαναγραφτεί πλήρως. Επίσης είναι δυνατό μέσα από κώδικα Rust να γίνει κλήση της C, σε περίπτωση που αυτό είναι απαραίτητο.

4.3 Package manager

Ένα σημαντικό κομμάτι της Rust είναι το Cargo, ο διαχειριστής πακέτων της (package manager), που χρησιμοποιείται σχεδόν πάντα στην γλώσσα. Με τη βοήθεια του είναι δυνατή η εύκολη επαναχρησιμοποίηση άλλων πακέτων και βιβλιοθηκών που είναι διαθέσιμα στο επίσημο αποθετήριο, το Crates.io. Επίσης γίνεται αρκετά ευκολότερη η διαχείριση της συμβατότητας με τις διάφορες εκδόσεις των εμπλεκόμενων προγραμμάτων, χάρη στις ενσωματωμένες λειτουργίες του Cargo.

Σχήμα 2: Το λογότυπο του Cargo[5]

Δεν αποτελεί κάποια πρωτοποριακή εφεύρεση της Rust, καθώς παρόμοιοι διαχειριστές προϋπήρχαν ήδη σε πολλές σύγχρονες γλώσσες υψηλότερου επιπέδου· ωστόσο σε σύγκριση με τη C ή τη C++ αποτελεί σημαντική εξέλιξη. Οι δύο τελευταίες γλώσσες δεν είχαν κάποιο επίσημο συγκεντρωτικό σύστημα διαχείρισης πακέτων. Εξαιτίας αυτού, η χρήση άλλων βιβλιοθηκών ήταν και είναι χειροκίνητη και κοπιαστική υπόθεση, αφού οι προγραμματιστές πρέπει να κατεβάσουν τις απαραίτητες βιβλιοθήκες από πολλές διαφορετικές ιστοσελίδες που έχουν τους δικούς τους κανόνες διανομής, να τοποθετήσουν τα απαραίτητα αρχεία στους κατάλληλους φακέλους, να ελέγξουν αν είναι συμβατά με τον κώδικα τους και να φροντίζουν οι ίδιοι για τις ενημερώσεις των βιβλιοθηκών, ακολουθώντας την ίδια διαδικασία.

4.4 Έλεγχοι κατά τη διαδικασία της μεταγλώττισης

Ο compiler της Rust είναι αρκετά πιο αυστηρός από αυτούς της C++, καθώς απαιτεί να διορθωθούν αρκετά λάθη ή προειδοποιήσεις κατά το στάδιο της μεταγλώττισης. Αυτό επιβαρύνει στιγμιαία τους προγραμματιστές, ωστόσο γλιτώνει λάθη τα οποία στη συνέχεια θα ήταν αρκετά δύσκολο να επιλυθούν. Χαρακτηριστικά παραδείγματα:

Χρειάζεται να γίνει έλεγχος κατά τη διάρκεια της μεταγλώττισης ώστε να βεβαιωθεί ότι το μοντέλο της ιδιοκτησίας (που αναφέρθηκε πιο πριν) ισχύει και δεν παρουσιάζει προβλήματα. Για παράδειγμα αν από την main καλέσουμε μια συνάρτηση test και της δώσουμε την τιμή testing_value (που δημιουργήθηκε στην main) δεν θα υπάρξει πρόβλημα. Ωστόσο αν δοκιμάσουμε να την ξανακαλέσουμε μια δεύτερη φορά ο compiler θα εμφανίσει σφάλμα, καθώς η main δεν είναι ο πλέον ο ιδιοκτήτης της testing_value, οπότε δεν μπορεί να την μεταβιβάσει. [8]

[5] Διαθέσιμο υπό άδεια Creative Commons Attribution *(CC-BY)*

4.5 Μικρού μεγέθους standard library

Η Rust έχει φροντίσει να κρατήσει το μέγεθος της κεντρικής βιβλιοθήκης μικρό. Αυτό γίνεται εφικτό χάρη στο Cargo, από το οποίο μπορεί κάποιος να κατεβάσει τις απαραίτητες εξαρτήσεις, καθιστώντας περιττό αυτές να είναι ενσωματωμένες στην κύρια εγκατάσταση.

Σε παλαιότερες γλώσσες, που δεν είχαν κάποιο διαχειριστή αρχείων ή δημιουργήθηκαν σε εποχές που η πρόσβαση στο διαδίκτυο δεν ήταν δεδομένη, ο σχεδιασμός τους περιελάμβανε μεγάλο αριθμό προεπιλεγμένων πακέτων. Ωστόσο, μετά από χρόνια, κάποια από αυτά έγιναν παρωχημένα και εμφανίστηκαν καλύτερες εναλλακτικές. Επειδή όμως τα παλιότερα πακέτα είναι ήδη διαδεδομένα, η αντικατάσταση τους από τα καινούργια είναι δύσκολη και πολλές φορές αναγκάζονται να συνυπάρχουν για μεγάλο χρονικό χρονικό διάστημα. [12]

Ο τρόπος λειτουργίας της Rust χρησιμεύει ώστε να μην έχει παρόμοια προβλήματα στο μέλλον. Επιπλέον προσφέρει το πλεονέκτημα πως το μικρό μέγεθος της κύριας βιβλιοθήκης της επιτρέπουν να λειτουργήσει σε ενσωματωμένα συστήματα με λιγοστό χώρο ή μνήμη.

5 Οικοσύστημα - Κοινότητα

Μιας και οι ρίζες της Rust ανάγονται στη Mozilla, σημαντικό δημιουργό έργων ανοιχτού κώδικα, η Rust αναπτύσσεται από τα πρώτα βήματα της με ανοιχτό και κοινοτικό τρόπο. Για να μην δημιουργηθεί μία κλειστή μονοκουλτούρα που ελέγχεται μόνο από μία εταιρεία, η Mozilla -παρότι συνεχίζει να συμμετέχει σημαντικά στην ανάπτυξη και χρηματοδότηση της γλώσσας- επέλεξε να μην χρησιμοποιήσει ηγετικά την επωνυμία της, από φόβο μήπως αποθαρρύνει την διάδοση της γλώσσας από άλλες εταιρείες. Το αποθετήριο της γλώσσας βρίσκεται στο Github και μπορεί να συμμετέχει οποιοσδήποτε εθελοντής ενδιαφέρεται. Τη στιγμή γραφής του κειμένου υπάρχουν περίπου 65.000 commits (αλλαγές στον κώδικα) από 1.800 άτομα.

Η κοινότητα της Rust επιδιώκει να είναι ιδιαίτερη προσιτή σε όσα άτομα επιθυμούν να συνεισφέρουν. [13] Γι' αυτό τον λόγο έχει υιοθετήσει έναν απλό *Κώδικα Δεοντολογίας (Code of Conduct)* [14] με σκοπό να αποτρέψει τις ανάρμοστες συμπεριφορές. [15, 16]

Μεγάλο μέρος της κοινότητας βρίσκεται και στο ομώνυμο subreddit *(reddit.com /r/rust)* στο οποίο εμφανίζονται σχεδόν όλα τα νέα που αφορούν τη Rust. Σε αντίθεση με άλλα subreddits τα οποία χρησιμοποιούνται κυρίως από χρήστες, στο συγκεκριμένο συχνάζουν πολλά από τα μέλη της ομάδας ανάπτυξης της Rust και συχνά μπορούν να αλιευθούν χρήσιμες πληροφορίες ή να λυθούν τυχόν απορίες. [17] Κάποια άλλα κοινοτικά projects γραμμένα σε Rust έχουν επιλέξει να δημιουργήσουν επίσης (μικρότερου μεγέθους) subreddits, όπως το */r/Redox* ή το */r/rust_gamedev*.[6]

5.1 Εμπορική υποστήριξη (εταιρείες / software)

Η Rust αρχίζει να χρησιμοποιείται από όλο και περισσότερες εταιρείες. Πέρα από τη Mozilla, το Dropbox έχει επενδύσει στη γλώσσα και την αξιοποιεί στις διαδικτιακές υποδομές που διαθέτει. Επίσης το Facebook είχε αναφέρει πως θα δημιουργήσει έναν server για το Mercurial, το σύστημα ελέγχου εκδόσεων (version control system) που χρησιμοποιεί για την ανάπτυξη του κώδικα του. Μία πλήρης λίστα των εταιρείων που χρησιμοποιούν τη γλώσσα είναι διαθέσιμη στην επίσημη ιστοσελίδα της Rust.

Το libsvrg είναι ένα από τα πρώτα λογισμικά ανοιχτού κώδικα χωρίς εμπορική υποστήριξη που ξεκίνησε να χρησιμοποιεί τη Rust. Επίσης το Tor, για το οποίο η ασφάλεια αποτελεί πρωτεύον ζήτημα, έχει αρχίσει τη συζήτηση για να ξαναγράψει τον κώδικα του σε μια γλώσσα που παρέχει ασφάλεια μνήμης.

[6] Το */r/playrust* ωστόσο δεν αναφέρεται στη γλώσσα προγραμματισμού, αλλά σε ένα άσχετο multiplayer παιχνίδι

5.2 Αξιόλογα software

Ένα πολύ ενδιαφέρον έμμεσο χαρακτηριστικό της Rust είναι η ποιότητα αρκετών λογισμικών που αναπτύσσονται στη γλώσσα. Οι περισσότερες υπάρχουσες βιβλιοθήκες ή προγράμματα σε άλλες γλώσσες είναι αποτέλεσμα πολλών χρόνων δουλειάς και προσπαθειών. Έτσι, από τη στιγμή που υπάρχουν έτοιμες λύσεις, συνήθως δεν αξίζει ο χρόνος για να ξαναγραφτούν κάποια προγράμματα με πιο σύγχρονες τακτικές που μπορεί να εμφανίστηκαν από τον καιρό της δημιουργίας τους.

Αντίθετα, επειδή η Rust αποτελεί παρθένο έδαφος, χρειάζεται να γραφτούν από την αρχή βιβλιοθήκες (ή άλλα προγράμματα) στην γλώσσα που να επιτελούν κάποιες βασικές εργασίες. Επειδή οι δημιουργοί αυτών δεν έχουν υπάρχον κώδικα να τους περιορίζει ή ενεργούς χρήστες που πιθανόν να δυσαρεστηθούν από τις αλλαγές, είναι ελεύθεροι να δοκιμάσουν νέους αλγόριθμους ή τεχνικές. Ως αποτέλεσμα, υπάρχει ένας μεγάλος αριθμός προγραμμάτων στην Rust που είναι πιο αποδοτικά από τις ήδη υπάρχουσες λύσεις.

5.2.1 Servo

Το 2013 η Mozilla, σε συνεργασία με τη Samsung, ξεκίνησε να αναπτύσσει στη Rust το Servo, μια μηχανή απεικόνισης διαδικτύου (web browser layout engine), η οποία εκμεταλλεύεται των χαρακτηριστικών της γλώσσας και μπορεί να χρησιμοποιήσει αποτελεσματικά πολυπύρηνους επεξεργαστές και τις όλο και πιο ισχυρές κάρτες γραφικών.

Το Servo βρίσκεται ακόμα υπό ανάπτυξη, καθώς δεν είναι εύκολο να δημιουργηθεί από το τίποτα ένα τόσο σύνθετο λογισμικό. Οι περισσότερες ιστοσελίδες εμφανίζουν προβλήματα απεικόνισης.

Επειδή όλες οι μηχανές απεικόνισης που χρησιμοποιούνται στους σημερινούς περιηγητές έχουν τις ρίζες τους στη δεκαετία του ’90, το Servo καταφέρνει στα περισσότερα benchmarks

Είναι αρκετά πιθανό στο -μακρινό- μέλλον το Servo να αντικαταστήσει το Gecko, την υπάρχουσα μηχανή απεικόνισης που χρησιμοποιείται στον Firefox. Επειδή ακόμα το Servo δεν έχει ολοκληρωθεί, η Mozilla έχει αρχίσει να ενσωματώνει στον Firefox κομμάτια του Servo που είναι ήδη έτοιμα και λειτουργικά, υπό το κωδικό όνομα Quantum. Παραδείγματα αποτελούν ο mp4 demuxer που υπάρχει από τον Firefox 53 και έπειτα.

5.2.2 Redox

Το Redox είναι ένα λειτουργικό σύστημα γραμμένο σε Rust. Δίνει έμφαση στην ασφάλεια και στοχεύει τα κρίσιμα ενσωματωμένα συστήματα. Για να το πετύχει αυτό χρησιμοποιεί την αρχιτεκτονική μικροπυρήνα (microkernel) και είναι αρκετά

μικρό έχοντας περίπου 50.000 γραμμές κώδικα, πολύ λιγότερο από τις εκατομμύρια γραμμές που έχουν άλλοι πυρήνες λειτουργικών συστημάτων. Παρότι γίνεται προσπάθεια να είναι δυνατόν να εκτελεστεί μεγάλος αριθμός προγραμμάτων που είναι γραμμένα για το Linux, δεν προσπαθεί να αντικαταστήσει το τελευταίο.

Σχήμα 3: Το λογότυπο του Redox

Πέρα από τον βασικό πυρήνα, στο Redox αναπτύσσονται και μερικά ακόμα λογισμικά, όπως το TFS, ένα σύστημα αρχείων επηρεασμένο από το ZFS, το Orbital, ένα παραθυρικό περιβάλλον, αλλά και το Ion, ένα κέλυφος για εκτέλεση εντολών.

6 Μειονεκτήματα της Rust

6.1 Δυσκολία εκμάθησης

Η Rust, όπως και άλλες γλώσσες που απευθύνονται στους τομείς που στοχεύει, έχει αρκετά μεγάλη δυσκολία εκμάθησης. Παρότι η Rust προσφέρει χαρακτηριστικά που μειώνουν τον πιθανό αριθμό των bugs και απρόοπτων συμπεριφορών, συνεχίζει να χρειάζεται αρκετή εξειδίκευση. Σε αντίθεση με γλώσσες πιο υψηλού επιπέδου, που παρέχουν αρκετές ευκολίες, ένας προγραμματιστής στην Rust πρέπει να έχει μια καλή κατανόηση της λειτουργίας της μνήμης και της αρχιτεκτονικής των υπολογιστών, καθώς και των διάφορων λογικών και συντακτικών χαρακτηριστικών της γλώσσας.

6.2 Υποστήριξη πληθώρας υπολογιστικών συστημάτων

Προς το παρόν η Rust υποστηρίζει μόνο τις πιο δημοφιλείς αρχιτεκτονικές επεξεργαστών, όπως είναι η x86 ή x86-64. Με την ενσωμάτωση της Rust στον Firefox αυτό άρχισε να δημιουργεί προβλήματα σε κάποιες διανομές Linux όπως το Debian ή το Fedora.

Πρέπει να αναφέρουμε ότι συνήθως αρκετές διανομές Linux είναι διαθέσιμες για παραπάνω από μία αρχιτεκτονικές και υποστηρίζουν πληθώρα συστημάτων. Επομένως η ξαφνική δυσκολία της μεταγλώττισης και διανομής ενός τόσο δημοφιλούς πακέτου όπως ο Firefox προκάλεσε αρκετές διαφωνίες. Σε περίπτωση που η Rust δεν καταφέρει μέσα στους επόμενους μήνες να υποστηρίξει αρκετές μικρότερες αρχιτεκτονικές, είναι πολύ πιθανόν κάποιες διανομές να σταματήσουν

να παρέχουν τον Firefox σε αυτές, να έχουν πεπαλαιωμένες εκδόσεις ή να αναγκαστούν σε ένα προσωρινό fork του συγκεκριμένου software.

Πέρα όμως από το πρόβλημα που προκύπτει λόγω της δημοφιλίας του Firefox, η μη υποστήριξη αρκετών αρχιτεκτονικών αποτελεί τροχοπέδη για μια γλώσσα που προσπαθεί να κερδίσει έδαφος έναντι της δημοφιλούς C, για την οποία υπάρχει σχεδόν παντού πλήρης υποστήριξη.

7 Πιθανή μελλοντική πορεία

Θα προσπαθήσω να αξιολογήσω την πιθανή μελλοντική πορεία της Rust λαμβάνοντας υπόψην πηγές που περιγράφουν εμπειρικά ή με δεδομένα την δημοτικότητα των γλωσσών προγραμματισμού. Έπειτα θα γίνει σύγκριση με το ερωτηματολόγιο ”State of Rust Survey 2016”, που διεξήχθη τον Μάϊο του 2016 (ένα χρόνο μετά την έκδοση 1.0) από τους δημιουργούς της Rust και στο οποίο απάντησαν 3.086 άτομα. Θα είχε πολύ ενδιαφέρον να γινόταν σύγκριση με τα αποτελέσματα της φετινής έρευνας, η οποία δέχθηκε (και συνεχίζει να δέχεται) πολύ περισσότερες απαντήσεις από την περσινή. Δυστυχώς τα αποτελέσματα θα ανακοινωθούν μερικές εβδομάδες μετά την λήξη της προθεσμίας συμμετοχής που είναι στις 12 Ιουνίου.

Θα ληφθούν επίσης υπόψη εμπειρικά ιστορικά στοιχεία όπως η πορεία γλωσσών με παρόμοιους στόχους με της Rust, όπως η Ada ή η D.

Παρακάτω παρουσιάζονται οι κυριότερες πληροφορίες που αλιεύθηκαν από το *“Empirical analysis of programming language adoption”*. [18] Αναφέρονται στοιχεία που αφορούν τη C και την C++, τους κυριότερους ανταγωνιστές της Rust.

- Το μεγαλύτερο ποσοστό αγοράς κατέχεται από γλώσσες γενικής χρήσης και όχι ειδικού σκοπού.
- Κάποιες λίγες γλώσσες χρησιμοποιούνται πολύ περισσότερο από όλες τις υπόλοιπες· οι 20 κορυφαίες γλώσσες χρησιμοποιούνται για το 95% των projects που υπάρχουν στο SourceForge.
- Οι προγραμματιστές μετακινούνται σε καινούργιες γλώσσες ανά ομάδες, αναλόγως με τις προηγούμενες γλώσσες που χρησιμοποιούν. Για παράδειγμα, ένα μικρό ποσοστό των ατόμων που χρησιμοποιούν Java γράφουν κώδικα και σε C ή C++, ενώ οι χρήστες C ή C++ είναι πιο πιθανόν να χρησιμοποιήσουν Perl από αυτούς της Java.
- Η δημοτικότητα των γλωσσών επηρεάζεται αρκετά από εξωτερικούς παράγοντες (μη τεχνικά χαρακτηριστικά) όπως οι διαθέσιμες βιβλιοθήκες ή η ήδη υπαρκτή εμπειρία των ατόμων.
- Οι μεγάλες εταιρίες έχουν διαφορετικές προτεραιότητες από τις μικρότερες εταιρείες ή όσους συνεισφέρουν σε ανοιχτό κώδικα. Η πρώτη κατηγορία ενδιαφέρεται περισσότερο για την διατήρηση του προυπάρχοντος κώδικα και την ασφάλεια ή σωστή λειτουργία του. Αντίθετα η δεύτερη ομάδα δίνει έμφαση στο οικοσύστημα ή την συμβατότητα με διάφορες πλατφόρμες και φαίνεται να είναι πιο πρόθυμη να υιοθετήσει καινούργιες γλώσσες.
- Κάθε γλώσσα έχει διαφορετικό χρόνο εκμάθησης που οι προγραμματιστές πρέπει να αφιερώσουν για να την κατανοήσουν. Η C και η C++ απαιτούν περισσότερη προσπάθεια από τις υπόλοιπες διαδεδομένες γλώσσες.

- Ο αριθμός γλωσσών που γνωρίζουν άτομα μικρότερης και μεγαλύτερης ηλικίας είναι παρόμοιος.
- Οι περισσότεροι ερωτούμενοι προγραμματιστές ενδιαφέρονται κυρίως για τις διαθέσιμες βιβλιοθήκες, την απόδοση και την δυνατότητα χρήσης χαρακτηριστικών αντικειμενοστραφών γλωσσών, ενώ ανέφεραν ότι στα διαθέσιμα χαρακτηριστικά μιας γλώσσας εκτιμούν την εκφραστικότητα και την δημιουργία κώδικα που μπορεί να επαναχρησιμοποιηθεί.

Αναφορές

[1] ***** Μαρακάκης Μ. (2014). *Prolog: Προγραμματισμός σε Λογική για Τεχνητή Νοημοσύνη*, Εκδόσεις Νέων Τεχνολογιών. Αθήνα, Ελλάδα.

[2] ***** Horowitz Ell. (1984). *Fundamentals of Programming Languages*, Springer Publishing. Manhattan, New York.

[3] Nørmark K. (2003). *Functional Programming in Scheme With Web Programming Examples*. Aalborg University.

[4] Hoare Gr. (2010). *Project Servo - Technology from the past come to save the future from itself*, Mozilla Annual Summit 2010. Whistler, Canada.

[5] Blandy J. & Orendorff J. (2016). Programming Rust, O'Reilly Media. Sebastopol, California.

[6] Dietz W., Li P., Regehr J. & Adve V. (2012). Understanding integer overflow in C/C++. In *34th International Conference on Software Engineering*. Zurich, Switzerland: IEEE Press, 760–770.

[7] Regehr J. (2011). *Race Condition vs. Data Race*. Embedded in Academia *https://blog.regehr.org/archives/490*

[8] Arlauskas N. (2015). *Explore the ownership system in Rust*. Ironic Blog *(https://nercury.github.io/rust/guide/2015/01/19/ownership.html)*

[9] Hoy J. (2016). *Rust's Ownership Model*. JackHoy.com *(http://www.jackhoy.com/web-applications/2016/11/10/rusts-ownership-model.html)*

[10] Bugaev S. (2016). *Understanding Rust: ownership, borrowing, lifetimes*. Medium *(https://medium.com/@bugaevc/understanding-rust-ownership-borrowing-lifetimes-ff9ee9f79a9c)*

[11] Hennessy J. L. & Patterson D. A. (2011). *Computer Architecture: A Quantitative Approach*, Elsevier. Amsterdam, The Netherlands

[12] † Rust Community (2017). *"Why does Rust's standard library feel so small?"* /r/rust *(https://www.reddit.com/r/rust/comments/6ddp3e/)*

[13] Anderson Br. (2017). *The Minimally-nice Open Source Software Maintainer*. brson.github.io *(https://brson.github.io/2017/04/05/minimally-nice-maintainer)*

[14] Rust Community. *The Rust Code of Conduct: https://www.rust-lang.org/conduct.html*

[15] † Rust Community (2013). *"A note on conduct (please read)"*. /r/rust *(https://www.reddit.com/r/rust/comments/1nvsdh/)*

[16] † Rust Community (2017). *“Question about Rust’s odd Code of Conduct”*. /r/rust *(https://www.reddit.com/r/rust/comments/6ewjt5/question_about_rusts_odd_code_of_conduct/didrult/)*

[17] Amirtha T. (2014). *Under The Hood Of Mozilla’s New Multi-Core Browser And The Open Source Language That Powers It*. FastCompany *(https://www.fastcompany.com/3027664/under-the-hood-of-mozillas-new-multi-core-browser-and-the-open-source-language-that-powers-i)*

[18] Meyerovich L. A. & Rabkin A. S. (2013). Empirical analysis of programming language adoption. *ACM SIGPLAN Notices*, 48(10), 1-18.

[19] Turner J. (2014). *State of Rust Survey 2016*. The Rust Programming Language Blog *(https://blog.rust-lang.org/2016/06/30/State-of-Rust-Survey-2016.html)*

[7] ***** Βιβλίο διαθέσιμο στην Κεντρική Βιβλιοθήκη του ΤΕΙ Ηπείρου

[8] † Συζήτηση με πολλούς συμμετέχοντες στο subreddit της Rust