Code Formatting

Εργαστήριο Λογισμικού - 255
Ορέστης Χιωτάκης

Code Formatting

Κάθε προγραμματιστής γράφει κώδικα με τον δικό του τρόπο. Κάποια παραδείγματα:

- Indentation με χρήση δύο ‘space’ **vs** με χρήση ενός ‘tab’
- if (statement == something) **vs** if (statement==something)
- struct list_node *nt **vs** struct list_node *next;
- int main(int argc, char **argv) {
  ….
  } **vs**
  Int main(int argc, char **argv)
  {
  ….
  }
- κτλ

Code Formatting

- Τώρα μπορείτε να φανταστείτε ένα μεγάλο πρότζεκτ (πχ. Linux Kernel, gcc, RocksDB κτλ) το οποίο εμπεριέχει εκατομμύρια γραμμές κώδικα μέσα σε χιλιάδες αρχεία, τα οποία έχουν γραφεί από χιλιάδες άτομα. Σκεφτείτε κάθε ένας από αυτούς τους προγραμματιστές να ακολουθεί όποιον τρόπο γραφής θέλει :) Αυτό θα επιφέρει αρκετά προβλήματα, όπως, αναγνωσιμότητας και συντήρησης του κώδικα από άλλους (μελλοντικούς) προγραμματιστές.
- Έτσι, κάθε πρότζεκτ και κάθε οργανισμός επιλέγει ένα **coding format**. Δηλαδή, ένα συγκεκριμένο στυλ γραφής κώδικα (κανόνες) που θα πρέπει να ακολουθεί κάθε μέλος του οργανισμού/κάθε εργαζόμενος πάνω στο πρότζεκτ.
- Υπάρχουν αρκετά εργαλεία (**formatters**) που μπορούν να μετατρέψουν οποιοδήποτε στυλ γραφής (κώδικα) σε ένα άλλο. Ένα τέτοιο εργαλείο είναι και το **clang-format** (llvm).

clang-format

clang-format - Εγκατάσταση

- **Ubuntu:** sudo apt install clang-format
- **Debian:** sudo apt install clang-format
- **Arch Linux:** sudo pacman -S clang (εγκατάσταση όλου του clang compiler*)
- **WSL:** sudo apt update; sudo apt install clang-format
- **Fedora:** sudo dnf install clang (*)
- **OS X:** sudo brew install clang-format

Μπορείτε να δείτε πληροφορίες και για άλλα linux-distro στο “command-not-found.com”

clang-format - Βασικά

Το clang-format έχει κάποια ήδη προκαθορισμένα στυλ γραφής κώδικα τα οποία μπορείτε να χρησιμοποιήσετε

- LLVM
- Google
- Mozilla
- Microsoft
- κ.α

Ειδοποιούμε το clang-format με την ρύθμιση `-style=xxxxx`. Δηλαδή, `clang-format -style=xxxx <files…>`

```
> clang-format -style=llvm main.c mystring.c
> clang-format -style=google src/db/database.c
```

clang-format - Βασικά

- Τώρα, τι γίνεται αν κάποιος δεν θέλει να χρησιμοποιήσει τα έτοιμα στυλ, αλλά, θέλει να φτιάξει ένα δικό του; Το clang-format μας παρέχει αυτή δυνατότητα επιλέγοντας -style=file, και ορίζοντας τους κανόνες μας μέσα σε ένα αρχείο εν ονόματι .clang-format. Όλα τα αρχεία που ξεκινούν με τελεία, είναι κρυφά αρχεία, δηλαδή, δεν φαίνονται με ένα απλό ls (ούτε και στο UI), παρά μόνο με ls -la.
- Το .clang-format αρχείο πρέπει να βρίσκεται στον parent-folder του πρότζεκτ. Αν βρίσκεται οπουδήποτε αλλού, τότε, πρέπει να δώσετε και το path όπου βρίσκεται, -style=file:format_file_path, (διαβάστε περισσότερες λεπτομέρειες εδώ).
- ```
  ~/hy255/            | > pwd
    .clang_format     | > ~/hy255/
    src/              | > clang-format -style=file src/*.c
      main.c          |
      something.c     |
  ```
  ```

clang-format - .clang-format επιλογές

- Το .clang-format αρχείο απαρτίζεται από διάφορες ρυθμίσεις της μορφής: **key: value**.

An example of a configuration file for multiple languages:

```
---
# We'll use defaults from the LLVM style, but with 4 columns indentation.
BasedOnStyle: LLVM
IndentWidth: 4
---
Language: Cpp
# Force pointers to the type for C++.
DerivePointerAlignment: false
PointerAlignment: Left
---
Language: JavaScript
# Use 100 columns for JS.
ColumnLimit: 100
---
Language: Proto
# Don't format .proto files.
DisableFormat: true
---
Language: CSharp
# Use 100 columns for C#.
ColumnLimit: 100
...
```

clang-format - .clang-format επιλογές

→ **AlignConsecutiveAssignments** (**BitFields**, **Declarations**, **Macros**)

```
AlignConsecutiveAssignments:
  Enabled: true
  AcrossEmptyLines: true
  AcrossComments: false
```

→ **Τιμές:** bool Enabled, bool AcrossEmptyLines, bool AcrossComments, bool AlignCompound, bool PadOperators.

→ Παράδειγμα

clang-format - .clang-format επιλογές

→ **AlignOperands**

→ **Τιμές:** true, false

→ Παράδειγμα

→ **ColumnLimit**

→ **Τιμές:** unsigned int

→ Παράδειγμα

Best Practises
Ο Κώδικας “εξηγεί” από μόνος του!

Best Practises

- Όταν ορίζουμε τύπους δεδομένων με *typedef*, τότε, βάζουμε στο τέλος του ονόματος "**_t**".
  typedef struct memory_buffer membuf_t
- Προτιμούμε να ονομάζουμε τους τύπους δεδομένων που φτιάχνουμε (typedefs, structs, enum) με όσο το δυνατόν πιο κατανοητά ονόματα γίνεται. Για παράδειγμα, έστω ότι θέλουμε να φτιάξουμε ένα struct που θα περιγράφει έναν δυναμικό πίνακα:

```
struct dynamic_array {
   size_t size;        // όχι 'sz' ή 's'
   size_t capacity;  // όχι 'cap'
   char *array;       // ή 'ptr'
}
```

Best Practices

- Δρούμε ομοίως και στις μεταβλητές που φτιάχνουμε στο πρόγραμμά μας. Για παράδειγμα:
  ```
  const char *filename;   // αντί για ‘fname’
  int new_line_counter;  // αντί για ‘nlcntr’ ή ‘counter’
  ```
- Χρήση των μηχανισμών της γλώσσας (όπως τα keyword) και του compiler για παραγωγή πιο ευανάγνωστου κώδικα. Για παράδειγμα:
  ```
  char *strcpy(char *restrict dest, const char *restrict src);
  ```
  - Το restrict keyword είναι απλά ένα hint προς τον χρήστη και τον compiler (optimizations) ότι όσοι pointer είναι τύπου restrict, δεν θα δείχνουν (dereference) στο ίδιο σημείο στη μνήμη.
  - Το const keyword ενημερώνει τον χρήστη και τον compiler ότι αυτή η συνάρτηση - δεν πρόκειται να τροποποιήσει το string “src”.
  - Τα παραπάνω είναι απλές ενδείξεις (hints) προς χρήστη (και compiler). Είναι όφελος του χρήστη να τα τα τειρεί!

Best Practises

- Χρήση **assert(...)**. Με την χρήση των assertion, πρακτικά εξηγούμε πως λειτουργούν οι συναρτήσεις σας και στους άλλους προγραμματιστές που θα διαβάσουν τον κώδικά σας. Για παράδειγμα, όταν κάποιος άλλος δει “assert(ptr == NULL)”, τότε θα είναι σίγουρος (χωρίς να διαβάσει σχόλια ή κάποιο documentation) ότι το “ptr” θα πρέπει να είναι οπωσδήποτε διάφορο του NULL. Κατά τη διάρκεια του compilation, μπορείτε να απενεργοποιήσετε όλα τα assert( ) δίνοντας το flag -DNDEBUG (σε gcc και σε clang) χωρίς να αφαιρέσετε τον κώδικα τον assert( ) από το πρόγραμμα!
- Τώρα, όπου χρειαστεί να εξηγήσουμε κάτι πιο περίπλοκο ή όταν κάτι είναι γενικά εκ φύσεως πιο δύσκολο, τότε, πρέπει να χρησιμοποιούμε ευανάγνωστα, μικρά, αλλά και περιγραφικά **σχόλια**.

Best Practises

- Δημιουργία περιγραφικού documentation (Doxygen). Δείτε το επόμενο slide για παράδειγμα!
- Χρήση function arguments, τα οποία είναι πάλι hints προς τους υπόλοιπους προγραμματιστές και τον compiler. Κάποια από τα attributes: warn_unused_result, nonnull, deprecated, noreturn. Για παράδειγμα:

```
extern int server_wait_threads(sHandle server_handle) __attribute__((deprecated("DO NOT USE!")));
```

- Τοποθετούμε συγκεκριμένες συναρτήσεις σε συγκεκριμένα αρχεία. Για παράδειγμα, όλες οι συναρτήσεις που τροποποιούν strings μέσα στο src/string.c, ενώ οι συναρτήσεις που τροποποιούν τα αρχεία μέσα στο src/file_handler.c. Οι αντίστοιχες βιβλιοθήκες μέσα στο φάκελο inc/ ή lib/ ή include/
- Σπάμε μεγάλα κομμάτια κώδικα σε συναρτήσεις (ή macros, δηλαδή, #define).

```
31
32  /**
33   * @brief Creates a new request object or updates an existing one. If @a req is equal to NULL ,then a new c_tcp_req object
34   * is created, otherwise @a req is updated with the provided (new) key and payload sizes. If @a rtype is @b REQ_GET, @b REQ_DEL,
35   * or @b REQ_EXISTS then @a paysz is ignored! If it is @b REQ_SCAN, then @a paysz acts as the range of the scan request.
36   * On success, of these two options, the former will return the newly allocated request object, while the latter will
37   * both return the updated request object and will set @a req to point to the updated object too. If the sum of @a keysz
38   * and @a paysz is less than the @a req current's sum, then no allocation occures. If it is greater, @a req is destroyed and
39   * a new request object is allocated. Check @b c_tcp_req_destroy() for more info on a request object's destruction.
40   * On failure, NULL is returned for both cases, @a req is neither destroyed nor points to another request object and
41   * @b errno is also set to indicate the error.
42   *
43   * @par ERRORS
44   * @b EINVAL  The request object @a req is not initialized, in case of an update
45   * @b ENOTSUP The request object's type @a rtype is not supported
46   *
47   * @param req request object
48   * @param rtype type of request
49   * @param keysz size of key
50   * @param paysz size of payload or length for SCAN request
51   * @return c_tcp_req
52   */
53  c_tcp_req c_tcp_req_factory(c_tcp_req *req, req_t rtype, size_t keysz, size_t paysz);
54
55  /**
56   * @brief Destroys the provided request object. On success, 0 is returned and @a req becomes invalid. As a result, any
57   * absolute pointers returned by either @b c_tcp_req_expose_key() or @b c_tcp_req_expose_payload() will become invalid too.
58   * On failure, -1 is returned and @b errno is set to indicate the error.
59   *
60   * @par ERRORS
61   * @b EINVAL The request object @a req is either equal to NULL, or not initialized
62   *
63   * @param req request object
64   * @return int
65   */
66  int c_tcp_req_destroy(c_tcp_req req);
```