Un langage machine pour débutants

sur machine virtuelle dédiée

Julien Bruguier

30 novembre 2013

Plan

Plan

Un ordinateur dans son plus simple appareil

Quels sont les éléments d'un ordinateur qu'on ne peut pas enlever ?

Un ordinateur dans son plus simple appareil

Quels sont les éléments d'un ordinateur qu'on ne peut pas enlever ?

- un **processeur**,

Un ordinateur dans son plus simple appareil

Quels sont les éléments d'un ordinateur qu'on ne peut pas enlever ?

- un **processeur**,
- une **mémoire**.

Un ordinateur dans son plus simple appareil

Quels sont les éléments d'un ordinateur qu'on ne peut pas enlever ?

- un **processeur**,
- une **mémoire**.

Il faudra donc s'intéresser de près à ces deux éléments pour écrire en langage machine.

Plan

Définition

Le langage machine est une **succession d'instructions** en mémoire **directement exécutées** par le processeur.

Définition

Le langage machine est une **succession d'instructions** en mémoire **directement exécutées** par le processeur.
Chaque instruction est câblée dans les circuits électroniques du processeur. On parle de **jeu d'instructions**.

Exemple de langage machine

Essayez juste de faire `$EDITOR $(which ls)` pour voir un binaire...

Exemple de langage machine

Essayez juste de faire `$EDITOR $(which ls)` pour voir un binaire...
Ca fait mal aux yeux, hein ?

Avantages et inconvénients

Avantages

Avantages et inconvénients

Avantages

1. Sur une vraie machine, c'est plutôt **rapide**,

Avantages et inconvénients

Avantages

1. Sur une vraie machine, c'est plutôt **rapide**,
2. les principes du langage machine **varient très peu** d'une machine à l'autre.

Avantages et inconvénients

Avantages

1. Sur une vraie machine, c'est plutôt **rapide**,
2. les principes du langage machine **varient très peu** d'une machine à l'autre.

Inconvénients

Avantages et inconvénients

Avantages

1. Sur une vraie machine, c'est plutôt **rapide**,
2. les principes du langage machine **varient très peu** d'une machine à l'autre.

Inconvénients

1. Une vraie machine, c'est complexe, ça demande des **connaissances techniques avancées**,

Avantages et inconvénients

Avantages

1. Sur une vraie machine, c'est plutôt **rapide**,
2. les principes du langage machine **varient très peu** d'une machine à l'autre.

Inconvénients

1. Une vraie machine, c'est complexe, ça demande des **connaissances techniques avancées**,
2. demande des **outils spécifiques**,

Avantages et inconvénients

Avantages

1. Sur une vraie machine, c'est plutôt **rapide**,
2. les principes du langage machine **varient très peu** d'une machine à l'autre.

Inconvénients

1. Une vraie machine, c'est complexe, ça demande des **connaissances techniques avancées**,
2. demande des **outils spécifiques**,
3. nécessite un **système d'exploitation** pour faire des applications utiles,

Avantages et inconvénients

Avantages

1. Sur une vraie machine, c'est plutôt **rapide**,
2. les principes du langage machine **varient très peu** d'une machine à l'autre.

Inconvénients

1. Une vraie machine, c'est complexe, ça demande des **connaissances techniques avancées**,
2. demande des **outils spécifiques**,
3. nécessite un **système d'exploitation** pour faire des applications utiles,
4. le code n'est **pas portable**.

Plan

Une machine dans un logiciel

Une machine dans un logiciel

Deux grands types de machines virtuelles :

Une machine dans un logiciel

Deux grands types de machines virtuelles :

1. des simulateurs de machines réelles : qemu, virtualbox,

Une machine dans un logiciel

Deux grands types de machines virtuelles :

1. des simulateurs de machines réelles : qemu, virtualbox,
2. des machines à part entière : JVM, **setlgg_machine**.

Plan

Plan

Pourquoi setlgg_machine ?

Pourquoi setlgg_machine ?

- C'est moi qui l'ai écrite ;)

Pourquoi setlgg_machine ?

- C'est moi qui l'ai écrite ;)
- c'est du **logiciel libre** (GPLv3),

Pourquoi setlgg_machine ?

- C'est moi qui l'ai écrite ;)
- c'est du **logiciel libre** (GPLv3),
- les outils : la machine et un éditeur de texte,

Pourquoi setlgg_machine ?

- C'est moi qui l'ai écrite ;)
- c'est du **logiciel libre** (GPLv3),
- les outils : la machine et un éditeur de texte,
- demande **peu de connaissances** pour commencer,

Pourquoi setlgg_machine ?

- C'est moi qui l'ai écrite ;)
- c'est du **logiciel libre** (GPLv3),
- les outils : la machine et un éditeur de texte,
- demande **peu de connaissances** pour commencer,
- extensible et directement exploitable pour des petites applications.

Architecture de la machine, vue d'avion

setlgg_machine est composée de quatre éléments :

Architecture de la machine, vue d'avion

setlgg_machine est composée de quatre éléments :

1. un **programme** pour stocker les instructions,

Architecture de la machine, vue d'avion

setlgg_machine est composée de quatre éléments :

1. un **programme** pour stocker les instructions,
2. une **mémoire** pour stocker les valeurs,

Architecture de la machine, vue d'avion

setlgg_machine est composée de quatre éléments :

1. un **programme** pour stocker les instructions,
2. une **mémoire** pour stocker les valeurs,
3. un **processeur** pour exécuter les instructions,

Architecture de la machine, vue d'avion

setlgg_machine est composée de quatre éléments :

1. un **programme** pour stocker les instructions,
2. une **mémoire** pour stocker les valeurs,
3. un **processeur** pour exécuter les instructions,
4. des **flux d'entrées/sorties** pour communiquer avec l'environnement,

Architecture de la machine, vue d'avion

setlgg_machine est composée de quatre éléments :

1. un **programme** pour stocker les instructions,
2. une **mémoire** pour stocker les valeurs,
3. un **processeur** pour exécuter les instructions,
4. des **flux d'entrées/sorties** pour communiquer avec l'environnement,
5. des plugins pour étendre la machine virtuelle.

Plan

Plan

Programme

Structure globale

Structure globale

- Un programme est une **séquence d'instructions**,

Structure globale

- Un programme est une **séquence d'instructions**,
- chaque instruction est accessible via sa position dans le programme : son **adresse programme**,

Structure globale

- Un programme est une **séquence d'instructions**,
- chaque instruction est accessible via sa position dans le programme : son **adresse programme**,
- la première instruction est à l'adresse <0>, la suivante à l'adresse <1>, etc...

Les instructions

Les instructions

- Une instruction est un **code** décrivant une **action**, avec des paramètres éventuels,

Les instructions

- Une instruction est un **code** décrivant une **action**, avec des paramètres éventuels,
- l'action modifie l'**état global de la machine**,

Les instructions

- Une instruction est un **code** décrivant une **action**, avec des paramètres éventuels,
- l'action modifie l'**état global de la machine**,
- l'état global étant l'**ensemble des valeurs** décrivant la machine à un **moment donné**.

Exemple

```
                :new INT/entier                          # <0>
                :read :stdin ~d LINE -> entier           # <1>
                :goto erreur :when @entier<0             # <2>
                @entier%10 -> entier                     # <3>
                :write :stdout ~s "Valeur : "            # <4>
                :write :stdout ~d @entier                # <5>
                :write :stdout ~s "\n"                   # <6>
                :delete entier                           # <7>
                :shutdown                                # <8>
    :label erreur
                :write :stderr ~s "Erreur\n"             # <9>
                :shutdown -1                             # <10>
```

Plan

Mémoire

Structure globale

Structure globale

- La mémoire est une **séquence de valeurs**,

Structure globale

- La mémoire est une **séquence de valeurs**,
- chaque valeur est accessible via une position : son **adresse mémoire**,

Structure globale

- La mémoire est une **séquence de valeurs**,
- chaque valeur est accessible via une position : son **adresse mémoire**,
- la première adresse mémoire est &0, la suivante est &1, etc...

Les valeurs

Les valeurs

- Les adresses peuvent être dans 3 états différents :

Les valeurs

- Les adresses peuvent être dans 3 états différents :
  1. **non allouées** : une lecture ou une écriture échouera,

Les valeurs

- Les adresses peuvent être dans 3 états différents :
  1. **non allouées** : une lecture ou une écriture échouera,
  2. **non initialisées** : une lecture échouera,

Les valeurs

- Les adresses peuvent être dans 3 états différents :
  1. **non allouées** : une lecture ou une écriture échouera,
  2. **non initialisées** : une lecture échouera,
  3. **initialisées** : les deux opérations sont autorisées ;

Les valeurs

- Les adresses peuvent être dans 3 états différents :
  1. **non allouées** : une lecture ou une écriture échouera,
  2. **non initialisées** : une lecture échouera,
  3. **initialisées** : les deux opérations sont autorisées ;
- les valeurs stockées sont **typées**, et des **tests de type** sont effectués à chaque lecture/écriture ;

Les valeurs

- Les adresses peuvent être dans 3 états différents :
  1. **non allouées** : une lecture ou une écriture échouera,
  2. **non initialisées** : une lecture échouera,
  3. **initialisées** : les deux opérations sont autorisées ;
- les valeurs stockées sont **typées**, et des **tests de type** sont effectués à chaque lecture/écriture ;
- les types gérés de base par la machine sont **INT** (entiers), **STR** (chaines de caractères), **BLN** (booléens), **PTR** (pointeurs) et **IOR** (références d'entrée/sortie).

Gestion de la mémoire

Gestion de la mémoire

- Le point noir du langage machine !

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,
- l'adressage mémoire est la partie complexe du langage machine sur cette architecture :

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,
- l'adressage mémoire est la partie complexe du langage machine sur cette architecture :
  - &33

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,
- l'adressage mémoire est la partie complexe du langage machine sur cette architecture :
  - `&33`
  - `:current`

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,
- l'adressage mémoire est la partie complexe du langage machine sur cette architecture :
  - &33
  - `:current`
  - globale

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,
- l'adressage mémoire est la partie complexe du langage machine sur cette architecture :
  - &33
  - `:current`
  - globale
  - &@pointeur

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,
- l'adressage mémoire est la partie complexe du langage machine sur cette architecture :
  - &33
  - `:current`
  - globale
  - &@pointeur
  - (pointeur + 17)

Gestion de la mémoire

- Le point noir du langage machine !
- La mémoire utilisée doit être allouée avant avec `:new` et libérée après avec `:delete`,
- l'adressage mémoire est la partie complexe du langage machine sur cette architecture :
  - &33
  - `:current`
  - globale
  - &@pointeur
  - (pointeur + 17)
  - ((&@(`:current` + 1) + @(globale + 4)) - 6)

Exemple

```
# Memory:
#> &0 : INT
#> &1 : STR "banzai"
#> &3 : PTR &1
#> &4 : IOR :stdout
#> &5 / booleen : BLN FALSE
#> &6 : INT 4
# Aliases:
# booleen -> &5
# Free addresses:
# From &2 on 1 addresses
# Free blocks:
# Block of 1 free addresses from &2
# End of memory
```

Plan

Processeur

Etat et registres

Etat et registres

- L'état du processeur est l'**ensemble des valeurs contenues dans ses registres**,

Etat et registres

- L'état du processeur est l'**ensemble des valeurs contenues dans ses registres**,
- un registre est une petite zone de mémoire située dans le processeur.

Etat et registres

- L'état du processeur est l'**ensemble des valeurs contenues dans ses registres**,
- un registre est une petite zone de mémoire située dans le processeur.
- Sur ce processeur, il existe seulement deux registres :

Etat et registres

- L'état du processeur est l'**ensemble des valeurs contenues dans ses registres**,
- un registre est une petite zone de mémoire située dans le processeur.
- Sur ce processeur, il existe seulement deux registres :
  1. un **registre pointeur d'instruction** : contient l'adresse programme de la *prochaine instruction à exécuter*,

Etat et registres

- L'état du processeur est l'**ensemble des valeurs contenues dans ses registres**,
- un registre est une petite zone de mémoire située dans le processeur.
- Sur ce processeur, il existe seulement deux registres :
  1. un **registre pointeur d'instruction** : contient l'adresse programme de la *prochaine instruction à exécuter*,
  2. un **registre pointeur de mémoire** : contient l'*adresse mémoire courante*, utilisée dans les appels de fonction pour accéder aux paramètres,

Etat et registres

- L'état du processeur est l'**ensemble des valeurs contenues dans ses registres**,
- un registre est une petite zone de mémoire située dans le processeur.
- Sur ce processeur, il existe seulement deux registres :
  1. un **registre pointeur d'instruction** : contient l'adresse programme de la *prochaine instruction à exécuter*,
  2. un **registre pointeur de mémoire** : contient l'*adresse mémoire courante*, utilisée dans les appels de fonction pour accéder aux paramètres,
- le pointeur d'instruction est à <0> au boot, et s'incrémente à chaque instruction,

Etat et registres

- L'état du processeur est l'**ensemble des valeurs contenues dans ses registres**,
- un registre est une petite zone de mémoire située dans le processeur.
- Sur ce processeur, il existe seulement deux registres :
  1. un **registre pointeur d'instruction** : contient l'adresse programme de la *prochaine instruction à exécuter*,
  2. un **registre pointeur de mémoire** : contient l'*adresse mémoire courante*, utilisée dans les appels de fonction pour accéder aux paramètres,
- le pointeur d'instruction est à <0> au boot, et s'incrémente à chaque instruction,
- le pointeur courant est modifié lors des appels de fonction et aux retours de fonction.

Pile d'états

Pile d'états

- L'état du processeur doit être sauvegardé lors d'un appel de fonction,

Pile d'états

- L'état du processeur doit être sauvegardé lors d'un appel de fonction,
- pour être restauré au retour de cet appel.

Pile d'états

- L'état du processeur doit être sauvegardé lors d'un appel de fonction,
- pour être restauré au retour de cet appel.
- L'état est **empilé** sur une pile avant d'être modifié, et **dépilé** pour être remis dans l'état sauvegardé.

Interruptions

Interruptions

- Une interruption est un **événement qui va perturber l'exécution** du programme :

Interruptions

- Une interruption est un **événement qui va perturber l'exécution** du programme :
  - une **interruption matérielle** — un signal système — sert à réagir à un stimulus externe,

Interruptions

- Une interruption est un **événement qui va perturber l'exécution** du programme :
  - une **interruption matérielle** — un signal système — sert à réagir à un stimulus externe,
  - une **interruption logicielle** — une exception dans une instruction — sert à adopter un comportement spécifique en cas d'erreur.

Interruptions

- Une interruption est un **événement qui va perturber l'exécution** du programme :
  - une **interruption matérielle** — un signal système — sert à réagir à un stimulus externe,
  - une **interruption logicielle** — une exception dans une instruction — sert à adopter un comportement spécifique en cas d'erreur.
- En cas d'interruption, par défaut, la machine s'arrête sur une erreur,

Interruptions

- Une interruption est un **événement qui va perturber l'exécution** du programme :
  - une **interruption matérielle** — un signal système — sert à réagir à un stimulus externe,
  - une **interruption logicielle** — une exception dans une instruction — sert à adopter un comportement spécifique en cas d'erreur.
- En cas d'interruption, par défaut, la machine s'arrête sur une erreur,
- mais une **fonction spéciale peut être appellée** pour traiter l'interruption.

Jeu d'instructions

Jeu d'instructions

1. Instructions de lecture/écriture mémoire, et des opérateurs basiques de calcul :
   - `->` `:new` `:delete` `:current` `:increase` `:decrease` `:clear`
   - `+` `-` `*` `/` `%` `=` `<>` `<` `>` `<=` `>=` `:and` `:or` `:xor` `:not` `:substring` `:size` `:rand`
   - `:date` `:convert`,

Jeu d'instructions

1. Instructions de lecture/écriture mémoire, et des opérateurs basiques de calcul :
   - `->` `:new` `:delete` `:current` `:increase` `:decrease` `:clear`
   - `+` `-` `*` `/` `%` `=` `<>` `<` `>` `<=` `>=` `:and` `:or` `:xor` `:not` `:substring` `:size` `:rand`
   - `:date` `:convert`,
2. instructions de contrôle de flot d'exécution :
   - `:goto` `:call` `:return` `:shutdown`, (`:label`),
   - `:wait` `:interruption` `:clone` `:child`,
   - `:when` `:unless` `:defined` `:initialized` `:empty` `:true` `:readable` `:writable` `:seekable`,

Jeu d'instructions

1. Instructions de lecture/écriture mémoire, et des opérateurs basiques de calcul :
   - `->` `:new` `:delete` `:current` `:increase` `:decrease` `:clear`
   - `+` `-` `*` `/` `%` `=` `<>` `<` `>` `<=` `>=` `:and` `:or` `:xor` `:not` `:substring` `:size` `:rand`
   - `:date` `:convert`,
2. instructions de contrôle de flot d'exécution :
   - `:goto` `:call` `:return` `:shutdown`, (`:label`),
   - `:wait` `:interruption` `:clone` `:child`,
   - `:when` `:unless` `:defined` `:initialized` `:empty` `:true` `:readable` `:writable` `:seekable`,
3. instructions d'entrée/sortie utilisant les flux :
   - `:open` `:close` `:read` `:write` `:seek` `:wait`.

Exemple

```
            :goto initialise :when entier :defined   # <0>
            :new INT*2/entier                        # <1>
:label initialise
            :read :stdin ~d LINE -> entier           # <2>
            :interruption FPE remplace_par_zero      # <3>
            :call inverse entier                     # <4>
            :write :stdout ~s "Valeur : "            # <5>
            :write :stdout ~ 5d @(entier + 1)        # <6>
            :write :stdout ~s "\n"                   # <7>
            :call inverse entier                     # <8>
            :shutdown                                # <9>
:label inverse
            1000 / @:current -> (:current + 1)       # <10>
            :return                                  # <11>
:label remplace_par_zero
            0 -> (:current + 1)                      # <12>
            :write :stderr ~s "Erreur\n"             # <13>
            :return 2                                # <14>
```

Plan

Flux d'entrée/sortie

Flux d'entrée/sortie

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,
- ils sont assimilables à des **tubes où transitent des données sérialisées**,

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,
- ils sont assimilables à des **tubes où transitent des données sérialisées**,
- une entrée dans la machine, une autre à l'extérieur.

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,
- ils sont assimilables à des **tubes où transitent des données sérialisées**,
- une entrée dans la machine, une autre à l'extérieur.
- Le support du flux peut être :

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,
- ils sont assimilables à des **tubes où transitent des données sérialisées**,
- une entrée dans la machine, une autre à l'extérieur.
- Le support du flux peut être :
  1. le **terminal**,

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,
- ils sont assimilables à des **tubes où transitent des données sérialisées**,
- une entrée dans la machine, une autre à l'extérieur.
- Le support du flux peut être :
  1. le **terminal**,
  2. un **fichier** ro, wo ou rw,

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,
- ils sont assimilables à des **tubes où transitent des données sérialisées**,
- une entrée dans la machine, une autre à l'extérieur.
- Le support du flux peut être :
  1. le **terminal**,
  2. un **fichier** ro, wo ou rw,
  3. une **connexion réseau** TCP ou UDP.

Flux d'entrée/sortie

- Les flux sont l'**interface d'échange avec l'extérieur**,
- ils sont assimilables à des **tubes où transitent des données sérialisées**,
- une entrée dans la machine, une autre à l'extérieur.
- Le support du flux peut être :
  1. le **terminal**,
  2. un **fichier** ro, wo ou rw,
  3. une **connexion réseau** TCP ou UDP.
- Les **références d'entrée/sortie** identifient les flux dans un programme. Elles sont indiquées dans les instructions modifiant un flux.

Plan

Plugins

Plugins

Plugins

- Les plugins sont des bibliothèques binaires avec des symboles particuliers,

Plugins

- Les plugins sont des bibliothèques binaires avec des symboles particuliers,
- ils peuvent **ajouter des instructions** au processeur,

Plugins

- Les plugins sont des bibliothèques binaires avec des symboles particuliers,
- ils peuvent **ajouter des instructions** au processeur,
- et permettre de **stocker en mémoire d'autres types** de données.

Plan

Exécution d'un programme

Exécution d'un programme

1. Chargement des plugins et vérification des dépendances inter plugins,

Exécution d'un programme

1. Chargement des plugins et vérification des dépendances inter plugins,
2. chargement du programme et vérification de la syntaxe,

Exécution d'un programme

1. Chargement des plugins et vérification des dépendances inter plugins,
2. chargement du programme et vérification de la syntaxe,
3. chargement de la mémoire et initialisation de la mémoire,

Exécution d'un programme

1. Chargement des plugins et vérification des dépendances inter plugins,
2. chargement du programme et vérification de la syntaxe,
3. chargement de la mémoire et initialisation de la mémoire,
4. **démarrage de la machine, exécution du programme et arrêt de la machine**,

Exécution d'un programme

1. Chargement des plugins et vérification des dépendances inter plugins,
2. chargement du programme et vérification de la syntaxe,
3. chargement de la mémoire et initialisation de la mémoire,
4. **démarrage de la machine, exécution du programme et arrêt de la machine**,
5. sauvegarde de la mémoire (succès) ou génération d'un coredump (erreur).

Exécution d'un programme

1. Chargement des plugins et vérification des dépendances inter plugins,
2. chargement du programme et vérification de la syntaxe,
3. chargement de la mémoire et initialisation de la mémoire,
4. **démarrage de la machine, exécution du programme et arrêt de la machine**,
5. sauvegarde de la mémoire (succès) ou génération d'un coredump (erreur).

Possibilité de lancer la machine en mode debug, pour une exécution pas à pas.

Plan

Fibonacci

$$fib(n) = \begin{cases} n & quand \quad n \in \{0,1\} \\ fib(n-1) + fib(n-2) & quand \quad n > 2 \end{cases}$$

```
        :new PTR/initial , INT, INT
        :convert ~d @input -> (initial+1)
        :call fibonacci initial
        :write :stdout ~d @(initial+2)
        :write :stdout ~s "\n"
        :delete initial*3
        :shutdown
    :label fibonacci # PTR iINT oINT
        :goto calcul_fib :when @(:current+1)>1
        @(:current+1) -> (:current+2)
        :return
    :label calcul_fib
        :new PTR , INT , INT , PTR , INT , INT -> :current
        @(:current+1) - 1 -> (&@:current+1)
        :call fibonacci &@:current
        @(:current+1) - 2 -> (&@:current+4)
        :call fibonacci (&@:current+3)
        @(&@:current+2) + @(&@:current+5) -> (:current+2)
        :delete &@:current*6
        :return
```

Un mini serveur HTTP

Un mini serveur HTTP

- Gestion des connexions TCP en mode serveur,

Un mini serveur HTTP

- Gestion des connexions TCP en mode serveur,
- Fork et gestion des chaines de caractères.

Un mini serveur HTTP

- Gestion des connexions TCP en mode serveur,
- Fork et gestion des chaines de caractères.
- Pour un serveur HTTP minimal, 50 instructions suffisent.

Un mini serveur HTTP

- Gestion des connexions TCP en mode serveur,
- Fork et gestion des chaines de caractères.
- Pour un serveur HTTP minimal, 50 instructions suffisent.
- Le site peut aussi être écrit en langage machine, la machine génère les pages à la volée.

Conclusion

- Des questions ? Posez-les, je ne mords — en général — pas. Pour les timides, il y a `mailto:projet.setlgg@pappy.tf`,
- Vous voulez tester ? L'archive est disponible sur `http://www.pappy.tf/projet/setlgg/index.php`.