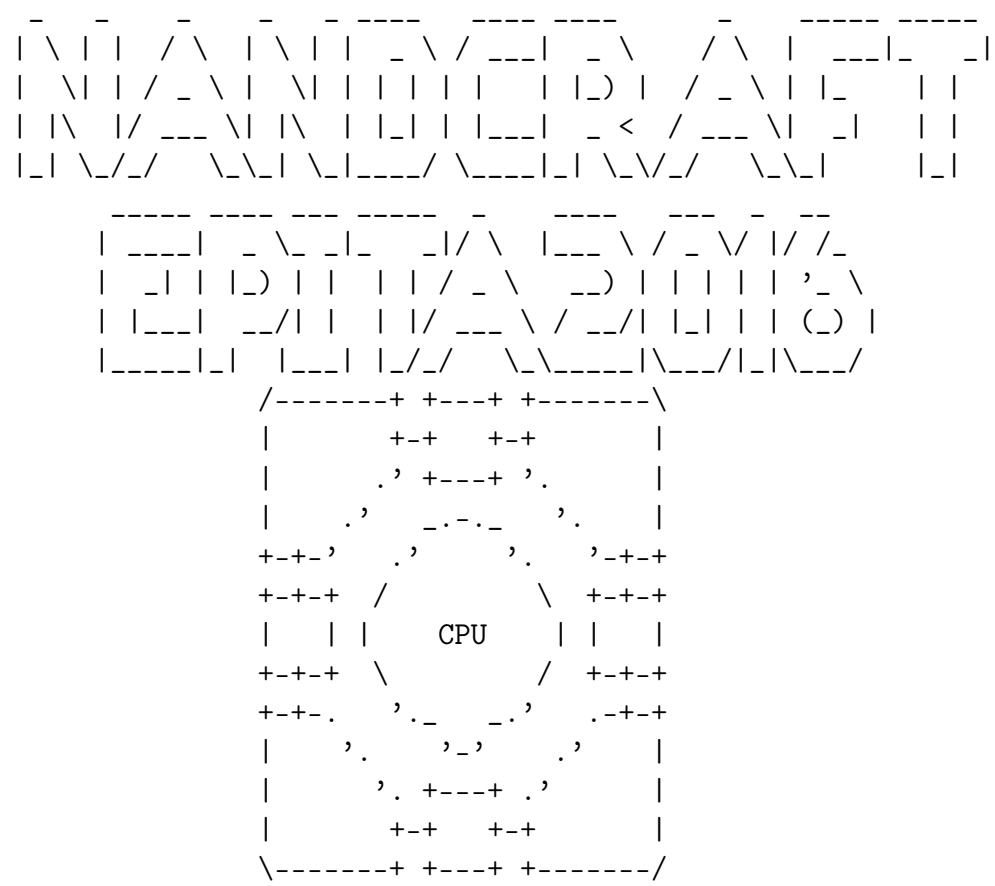
CPU - Cube Processing Unit
audebe_r - halfr
hervot_p - Dettorer
eddequ_n - nass
pruvot_a - Chaf

---

tl;dr On va créer un ordinateur, c'est un super projet.



Sommaire

1 Présentation du groupe

1.1 Amaury *Chaf*

Ma passion pour l'informatique est apparue dès mon plus jeune âge en parcourant de nombreux jeux vidéo. Contrairement à beaucoup de personnes à l'Epita, je ne suis pas venu pour créer des jeux vidéo plus tard, j'ai toujours voulu explorer la sécurité réseau. C'est pourquoi cette année j'ai rejoins un groupe ayant un projet atypique mais néanmoins très attrayant : quoi de mieux que de fabriquer un ordinateur de A à Z pour comprendre son fonctionnement et pouvoir mieux le protéger ?

1.2 Nassim *nass*

L'informatique m'a sauvé ! Sans elle, je serai très certainement entrain de dealer de la drogue, en prison ou bien mort dans une rixe de gangs. Je lui dois la vie à tout jamais...

Plus sérieusement, c'est avec plaisir que j'ai voulu participer à ce projet bien spécial qu'est NANDCRAFT. Toujours passionné par ce milieu qu'est l'informatique mais également l'électronique, la réalisation d'un jeu m'est apparue bien trop restrictive étant donné que j'en avais déjà codé en C. Accompagné de trois personnes très motivées et travailleuses, je pense pouvoir faire évoluer très rapidement mes connaissances tant en programmation impérative et fonctionnelle que sur la manière dont fonctionne un ordinateur. J'attends énormément de ce projet encore une fois sur le plan de la programmation mais également relationnel car le développement en équipe nécessite des capacités qu'il me faudra acquérir.

1.3 Paul *Dettorer*

Je suis arrivé à EPITA sans avoir la prétention d'être un passioné, mais avec la certitude que beaucoup de domaines de l'informatique allaient m'intéresser.

N'ayant que touché en surface à plusieurs choses différentes comme l'algorithmique, le réseau, le multimédia ou les systèmes Unix, créer un jeu vidéo m'aurait autant intéressé qu'un tout autre type de projet. Cependant je vois en NANDCRAFT l'occasion d'en apprendre bien plus sur le fonctionnement d'un ordinateur et de son système d'exploitation.

J'avais rencontré Rémi pendant les vacances d'été, je passe mes soirées en salle machine avec Nassim depuis le séminaire et je m'entends bien avec Amaury. Je sais que nous partageons beaucoup d'intérets et que nous prendrons beaucoup de plaisir à travailler sur ce projet.

tl;dr : J'aime les chats

1.4 Rémi *halfr*

Je suis venu à EPITA car je pense que pour prétendre être un ingénieur en informatique compétent il est nécessaire de s'investir pleinement, sans compter les heures, ni les bols de café.

J'ai beaucoup trop joué aux jeux vidéos pour pouvoir en coder un, je pense que c'est une source infinie de frustration et d'échecs. C'est pourquoi j'ai proposé à mon groupe la création d'un ordinateur en OCaml. Pour leur prouver la faisabilité de ce projet je leur ai

montré un bout de code que j'avais écrit il y a quelques années. Il était de conception simple et arrivait à simuler un CPU en se basant uniquement sur la porte logique `NAND`.

J'ai rencontré Dettorer avant même d'être entré à EPITA. Nous nous sommes instantanément bien entendus et c'est avec plaisir que je lui ai proposé de rejoindre le projet. De même pour Nass, avec qui nous parlions d'idées de projet dès Août. Quant à Chaf, nous sommes dans la même classe, nous avons sympathisé et ainsi fut formé notre groupe de projet.

Ils sont tous géniaux, je les aime.

> *tl;dr Un jeune connard de geek. Ne l'invitez jamais en soirée. C'est un troll, c'est comme ça. Personne ne l'aime et nous pensons qu'il a insulté ta mère.*
>
> – Fichier des Renseignement Généraux d'une planète lointaine

2 Présentation du projet

2.1 NANDCRAFT

What I hear, I forget; What I see, I remember; What I do, I understand.

– Confusius, 551-479 av. J.-C.

Il y a fort longtemps, tous les ingénieurs en informatique comprenaient parfaitement le fonctionnement des ordinateurs. L'ensemble des intéractions entre la machine, les logiciels, les compilateurs et le système d'exploitation étaient simples et suffisament transparents pour pouvoir comprendre l'ensemble des opérations de l'ordinateur. Mais la technologie a évoluée, elle est devenue de plus en plus complexe et cette simplicité a été perdue. Les fondements de l'informatique, l'essence de la discipline, sont maintenant cachés par des interfaces obscures et des implémentations propriétaires.

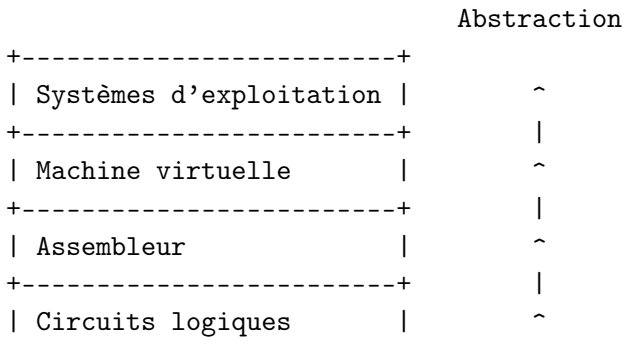
C'est pourquoi nous allons construire un ordinateur en partant d'un élément atomique de logique : la porte `NAND`. Avec elle nous construiront une machine capable d'exécuter du code stocké dans une mémoire interne (elle même créée à l'aide de la porte `NAND`). Puis nous créerons un autre language qui facilitera l'écriture du code machine. D'abstractions en abstractions, on crée des langages toujours plus simples à utiliser. Chaque langage se base ainsi sur le précédent.

Dans le cadre de notre projet chaque membre du groupe travaille sur une partie différente de l'abstraction. Elle est uniquement dépendant de la partie directement inférieure à elle. Ce sera l'occasion de s'entraider et de profiter réellement de la dimension humaine du projet. La diversité dans les projets implique une importante source de découverte et de partage entre les membres du groupe. En effet, malgré l'indépendance des projets, nous devrons toujours nous assurer de la cohérence globale du système.

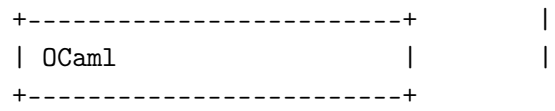
Nous avons choisi d'utiliser le langage OCaml pour la réalisation de `NANDCRAFT` car des nazis nous l'ont ordonné dans un songe[1]. En effet, nous allons principalement manipuler des structures de données sous forme de graphes ainsi que des langages, ce pourquoi OCaml est très adapté.

Ce projet nous a été inspiré par la lecture de *The Elements of Computing Systems* qui décrit un projet similaire.

2.1.1 Schéma global du projet



[1] En réalité c'est faux, car nous ne dormons pas.



2.2 Portée du projet

Nous avons répertorié les domaines que nous allons aborder :

Bas niveau Portes logiques, arithmétique booléenne, multiplexeurs, flip-flop, registres, RAM, compteurs binaires, `Hardware Description Language` (`HDL`), simulation et test de circuits.

Architecture ALU/CPU, langage machine, assembleur, mode d'addressage, entrées/sorties liées à la mémoire.

Système d'exploitation Gestion de la mêmoire, bibliothèque mathématique, gestion de l'écran, entrées/sorties sur un fichier, langage de haut niveau.

Langages de programmation Orienté-objet, types abstraits, portée des symboles, syntaxe et sémantique.

Compilateurs Analyse lexicale, décomposition analytique progressive, tables de symboles, machine à pile, génération de code, implémentation des tableaux et des objets.

Structures de données et algorithmes Piles, tables de hachage, listes, allocation mémoire.

2.3 Sous-projets

> *Everything should be made as simple as possible, but not any simpler.*
>
> – Albert Einstein

Par la suite nous utiliserons les abbréviations suivantes :

TECS *The Elements of Computing Systems* par Noam Nisan et Shimon Schocken

SCO *Structured Computer Organization* par Andrew S. Tanenbaum

LLC *Le Langage Caml* par Pierre Weis et Xavier Leroy

2.3.1 Niveau logique :Chaf:

- Description
  Les portes logiques sont les éléments de base de l'ensemble des appareils numériques. Cette partie est la plus bas-niveau du projet, elle correspond à la simulation de circuits logiques qui pourraient être fabriqués physiquement. Le comportement de la machine sera donc imité grâce aux travaux réalisés dans cette partie de NANDCRAFT.

  Les références dans TECS à propos de ce sujet :

  Chapitre 1 Boolean Logic

  Chapitre 2 Combinatorial Chips

  Chapitre 3 Sequential Chips

  ainsi que celles dans SCO :

  Chapitre 3 THE DIGITAL LOGIC LEVEL

- Palier 0 - Logique combinatoire
  `DEADLINE:` Soutenance 1

  > *Such simple things, And we make of them something so complex it defeats us, Almost.*
  > – John Ashbery, poête américain

  Premièrement, il est nécessaire d'implémenter les différentes portes logiques utilisées dans la fabrication de n'importe quel système numérique aujourd'hui. Comme le nom du projet l'indique, nous partirons de la seule porte `NAND` pour construire toutes les autres.

- Palier 1 - Unité arithmétique et logique (UAL)
  `DEADLINE:` Soutenance 2

  > *Counting is the religin of this generation, its hope and salvation.*
  > – Gertrude Stein (1874-1946)

  Nous aurons également besoin d'une UAL qui permettra d'effectuer un certain nombre d'opérations arithmétiques. Nous construirons ici une UAL composée d'un demi-additionneur (permettant l'addition de deux bits), d'un additionneur (permettant l'addition de trois bits) et d'un additionneur complet (permettant l'addition de deux entiers de n-bits). En effet, la plupart des opérations nécessaires peuvent se rapporter à une suite d'additions (nous remarquerons par exemple que `x - y = x + (-y)`).

  Nous écrirons ensuite un outil permettant de créer et d'éditer des circuits logiques combinatoires et de les tester.

  Note :

  – L'objectif de ce sous-projet est de manipuler le circuit de façon symbolique, cet aspect ne doit pas être négligé, même à ce palier.

Référence :

- Dans SCO, l'*Annexe A: BINARY NUMBERS* fournit des informations sur la repésentation des nombres en binaires.

- Palier 2 - Logique séquentielle
`DEADLINE:` Soutenance 3

  Time is an illusion. Lunchtime doubly so.
  – Ford Prefect

  Nous avons maintenant un système capable de calculer des valeurs, Cependant nous avons besoin de pouvoir stocker et réutiliser ces valeurs grâce à de la mémoire. Nous allons donc construire une horloge pour représenter le temps, des bascules et des registres pour pouvoir utiliser des valeurs dépendant de leur état précédent et les stocker, de la mémoire vive où stocker ces valeurs et des compteurs pour incrémenter/décrémenter le temps.

  Il sera aussi nécessaire de créer un outil permettant des créer, éditer et tester des circuits logiques séquentiels.

- Palier 3 - Contruire la machine
`DEADLINE:` Soutenance 4

  L'intelligence humaine commence quand l'homme fabrique des outils à faire des outils.
  – Henri Bergson (1859-1941)

  Nous disposons d'un circuit logique capable d'effectuer des opérations arithmétiques ainsi que d'un ensemble de registres dont la valeur est liée au temps.

  Il ne reste plus qu'à créer un circuit unissant ces deux composants qui aurait pour fonction l'execution de code binaire stocké dans la mémoire. Il faut pour cela conserver la position dans le programme puis faire varier cette valeur en fonction du code : à chaque instruction executée on incrémente le pointeur programme, à l'exception de certaines intructions dont le rôle est spécifiquement de se déplacer dans le programme.

  Le code binaire généré par l'assembleur est chargé dans la mémoire morte de la machine. Suite à cela le circuit est lancé, ce qui signifie que les opérations s'opèrent, changeant les valeurs de la ram et ainsi se comportant comme un véritable ordinateur.

- Palier 4 - Bonus : FPGA
  Nous écrirons un convertisseur de circuit symbolique vers verilog/vhdl puis exécuterons ce code sur une carte d'expérimentation dotée d'un Field Programmable Gate Array (FPGA). Un FPGA est un composant reprogrammable capable de se compoter comme un circuit logique donné par l'utilisateur.

2.3.2 Assembleur :nass:

- Qu'est-ce qu'un assembleur ?
  L'assembleur est l'outil qui gère la traduction de programmes en langage symbolique assembleur vers le langage machine. L'assembleur s'occupe également de la gestion des symboles définis par le système et l'utilisateur et leur assigne à des adresses de la mémoire physique selon les besoins.

- Description
  Les articles Wikipédia relatifs à ce sujet :

  – Assembly language
  – Instruction set ou Instruction Set Architecture (ISA)

  Les références à cette partie du projet dans TECS :

  Chapitre 4 Machine Language

  Chapitre 5 Computer Architecture

  Chapitre 6 Assembler

  La référence dans SCO, muni d'un exemple générique d'assembleur :

  Chapitre 7 THE ASSEMBLY LANGUAGE LEVEL

  Annexe C ASSEMBLY LANGUAGE PROGRAMMING

  Dans LLC :

  Chapiter 14 : Simulation d'un processeur présente l'intégralité des étapes nécessaires pour créer l'assembleur et l'émulateur en OCaml.

- Palier 0 - Assembleur
  `DEADLINE:` Soutenance 1

  > *There are only 10 types of people in the world: those who understand binary, and those who don't.*

  Afin de nous simplifier la tâche vis à de la création de l´assembleur, un assembleur en deux passes semble le plus adapté. En effet, cet assembleur lit deux fois de bout en bout le code qui lui est transmis.

  Nous commençons par initialiser une table des symboles avec 16 symboles prédéfinis respectivements associés à leur adresse mémoire dans la RAM.

  Durant la première passe, aucun code n´est généré. Le but de cette étape est de créer une table de symboles associés à leur valeur.

  Durant la seconde, on analyse le code reçu puis détermine, lorsque l´instruction reçue est de type symbolique, si le symbole est déjà dans la table. Si tel est le cas, il est remplacé par sa valeur sinon on l´ajoute à la table de symboles associé à la prochaine

adresse mémoire disponible dans la RAM (qui démarre à 16, les 15 premiers étant réservés aux symboles prédéfinis lors de l´initialisation).

Référence :

- Dans SCO, *Section 7.3: THE ASSEMBLY PROCESS* décrit l'implémentation d'un assembleur en deux étapes.

- Palier 1 - Émulateur
`DEADLINE`: Soutenance 2

  If everything seems under control, you're just not fast enough.
  –Mario Andretti, race car champion

  Nous comptons écrire un émulateur permettant d'exécuter le binaire produit par l'assembleur.

- Palier 2 - Débuggeur
`DEADLINE`: Soutenance 3

  L'intêret du débuggeur sera essentiellement de pouvoir récupérer l'état du système et éventuellement de le modifier.

  Note :

  - Il sera judicieux de gérer des informations de débuggages fournie par les projets supérieurs. Il faudra gérer les informations reçues par le bytecode afin de faciliter le débuggage. Exemple : nom de la procédure, nom, taille et type des symboles.

- Palier 3 - Écran et Entrées/sorties
`DEADLINE`: Soutenance 4

  I can't get no satisfaction.
  –Mick Jagger

  Gérer l'écran dans l'émulateur nous semble être une tâche très importante du fait de l'aspect graphique. L'écran en question sera monochrome avec pour correspondance les bits à 1 pour les pixels noirs et à 0 pour les pixels blancs.

  Gérer les entrées/sorties dans l'émulateur est également extrêmement important car nous avons besoin d'un clavier pour une interaction correcte entre l'utilisateur et notre machine.

- Palier 4 - Bonus éventuels : linker et loader
  Afin d'optimiser le fonctionnenent de l'assembleur et du binaire, nous essayerons d'inclure un linker dynamique et un loader. Ceux-ci auront plus précisement pour but de gérer plusieurs fichiers et charger des binaires à la volée.

  Références : Dans SCO : *Section 7.4 LINKING AND LOADING* décrit le fonctionnement d'un loader et d'un linker.

  Il existe de la documentation très détaillée sur les linkers et les loaders : http://www.iecc.com/linker/

- Palier 5 - Bonus éventuellement éventuel : désassembleur
  Nous tenterons de coder si le temps le permet un désassembleur qui permet de retrouver le code assembleur à partir du code machine.

2.3.3 Machine virtuelle :Dettorer:

- Description

  > *Programmers are creators of universes for which they alone are responsible. Universes of virtually unlimited complexity can be created in the form of computer programs.*
  > – Joseph Weizenbaum

  La machine virtuelle est un traducteur intermédiaire entre le langage de haut-niveau et l'assembleur. Elle permet l'exécution d'un programme dans un langage donné sur n'importe quelle machine en ne nécessitant que très peu de modifications, voir aucune. Le bytecode généré par le compilateur est donc prévu pour fonctionner sur la machine virtuelle, qui le traduira en code assembleur simplifié et compatible avec la machine cible.

  C'est la méthode choisie par Microsoft pour .NET et par java avec la Java Runtime Machine.

  Wikipedia :

  - Stack machine
  - Virtual machine
  - p-code machine

  Références dans TECS :

  Chapitre 7 Virtual Machine I: Arithmetic

  Chapitre 8 Virtual Machine II: Control

  Ailleurs :

  - Writing an Interpreter with PyPy

- Palier 0 - Arithmétique
  `DEADLINE:` Soutenance 1

  La première étape consiste à traduire les opérations simples telles que l'addition et la soustraction via un système de pile. Il faudra pour cela savoir gérer les accès mémoire.

- Palier 1 - Contrôle du flux
  `DEADLINE:` Soutenance 2

  Il faut ensuite gérer les instructions de type goto, qui permettent l'implémentation des conditions et des boucles. À partir de cela il sera possible de traduire des appels de fonction au sein du flux d'exécution.

- Palier 2 - Émulateur
  `DEADLINE:` Soutenance 3

  Écrire un émulateur capable d'interpréter le codé écrit en langage intermédiaire sera la prochaine étape.

- Palier 3 - Bytecode binaire
  `DEADLINE:` Soutenance 4

  Enfin, définir une ISA afin de représenter le bytecode sous forme binaire et non sous forme de texte permettra un grand gain de place.

- tl;dr
  La machine virtuelle marche à pile. Elle est une étape dans la traduction du code en langage machine et augmente sa portabilité.

2.3.4 Compilateur :halfr:

- Description
  Voici la dernière phase d'abstraction du projet. Nous allons définir un langage de haut-niveau utilisable pour écrire le système d'exploitation tout en étant suffisamment simple pour ne pas compliquer la phase de compilation.

  Articles wikipedia :

  - Compiler
  - High-level programming language

  Références dans TECS :

  Chapitre 9 Programming Language

  Chapitre 10 Compiler I: Syntax Analysis

  Chapitre 11 Compiler II: Code Generation

  Dans LLC :

  Chapiter 15 Compilation de mini-Pascal

  Ailleurs sur internet :

  - Let's Build a Compiler, original turbo pascal version
  - Let's Build a Compiler, forth, more recent
  - Implementing a language with LLVM, ocaml version
  - Why ML/OCaml are good for writing compilers

- Palier 0 - Analyse syntaxique
  `DEADLINE`: Soutenance 1

  On commence par analyser le code source pour vérifier sa syntaxe et générer une représentation interne sous forme d'arbre.

  L'objectif de ce palier est d'afficher cet arbre sous la forme d'une image en utilisant le langage dot.

  Références :

  - Interprète BASIC dans DO-OCAML
  - Implementing Scheme in OCaml, part 1 et part 2

- Palier 1 - Génération de code
  `DEADLINE`: Soutenance 2

  Une fois l'AST généré, nous produirons le code équivalent en langage intermédiaire qui est ici celui développé dans le cadre du sous-projet. `Machine Virtuelle`.

- Palier 2 - Bibliothèque standard
  `DEADLINE`: Soutenance 3

L'ASCII m'a tuer.

Maintenant que l'on dispose d'un langage et d'un compilateur, il faut le peupler de fonctions utiles.

On implémentera les fonctions suivantes :

- manipulation des chaînes de caractères ;
- fonctions mathématiques ;
- gestion des tableaux de taille variable ;
- entrées/sorties basiques.

- Palier 3 - Système d'exploitation
  `DEADLINE:` Soutenance 4

  Finalement on codera un ensemble d'éléments formant les couches supérieures de l'ordinateur :

  - une bibliothèque graphique permettant d'afficher des formes géométriques simples ;
  - une bibliothèque pour écrire du texte à l'écran ;
  - des utilitaires basiques tels une ligne de commande ou une calculatrice.

- tl;dr
  Ça va être dur, il va falloir coder. Mais en prenant les bons outils et ne visant pas trop haut on peut obtenir un système qui roxxx.

3 Considérations annexes

3.1 Calendrier

Projet	Soutenance 1	Soutenance 2	Soutenance 3	Soutenance FINALE
Logique par **Chaf** et *nass*	Palier 0	Palier 1	Palier 2	Palier 3
Assembleur par **nass**, *Chaf* et *Dettorer*	Palier 0	Palier 1	Palier 2	Palier 3
Machine Virtuelle par **Dettorer**, *nass* et *halfr*	Palier 0	Palier 1	Palier 2	Palier 3
Compilateur par **halfr** et *Dettorer*	Palier 0	Palier 1	Palier 2	Palier 3
Awesomeness	25%	50%	100%	OMFG!!!!1!!1!!

(nous ne sommes pas des robots.)
La personne en **gras** est le « chef du sous-projet ».
Cette page est intentionnellement laissée blanche, parce que.

3.2 Budget

Types de dépense	nass	halfr	Dettorer	Chaf	Unités
coiffeur	29	5000	2500	8	euro
bouquin TECS	22	22	22	22	dollar
chats	1	1	37	1	chat
pin's anti-bocal	42	43	42	42	TIG
costumes (nazis)	100	100	100	100	hitler
costumes (emo)	15	15	15	15	suicide
barbecue	evolutek	evolutek	evolutek	evolutek	sildra
conférences	12	13	14	12	delroth
code	5015	SEGFAULT	4096	4870	chameau
boissons	coca zero	fanta orange	ice tea	dr.pepper	Kilosucre
déodorant	beaucoup	énormement	PFIOUUU	ZOMGGG	sueur
Doctor	wibbly	wobbly	timey	wimey	stuff
??????????	Questions	Remarques	Soucis	Choses	pas claires
Bullshit	4	4	4	4	Soutenances
Plaisir	45 (fake)	0	0	0	Femmes/Hommes
NYAN	NYAN	NYAN	NYAN	NYAN	NYAN
Fedora	0	0	0	0	euro
Awesome	0	0	0	0	euro
Vim	0	0	0	0	euro
Emacs	0	0	0	0	euro
Org-mode	0	0	0	0	euro
OCaml	0	0	0	0	euro
Mercurial	0	0	0	0	euro
zlock kikoo	0	0	0	0	euro
netsoul	0	0	0	0	euro
Ouverture Culturelle	0	0	0	0	/20
TOTAL	25%	25%	25%	25%	NANDCRAFT

4 Conclusion

Ce projet nous apportera une meilleure connaissance du monde informatique ainsi que du fonctionnement des ordinateurs.

```
tl;dr On est très heureux.
```