UNIVERSITÉ PIERRE ET MARIE-CURIE

MASTER 1
PROJET STL

php_ of_ OCaml

réalisé par :

Hacene KASDI - 3524196
Sofiane GHERSA - 3525755

travail encadré par :

M. Vincent Botbol
M. Abdelraouf Ouadjaout

Année universitaire : 2016-2017

Table des matières

Introduction

Le monde de compilation a grandement changé, les langages de programmation sont en perpétuel évolution. Avant qu'un programme puisse être lancé il faut qu'il soit traduit dans une forme qu'un ordinateur puisse l'exécuter. Les logiciels qui réalisent cette traduction sont des compilateurs.

Notre projet consiste à développer un compilateur *php_of_OCaml* en partant du compilateur OCaml, permettant aux développeurs de ce langage fonctionnel de traduire du code écrit en OCaml vers un code lisible en langage PHP, interprété et typé dynamiquement, qui sera destiné à être exécuté sur les serveurs web.

OCaml est un langage fonctionnel augmenté de traits, permettant la programmation impérative et typé statiquement. Il se prête à la programmation dans un style fonctionnel, impératif ou orienté objet. Pour toutes ces raisons, OCaml entre dans la catégorie des langages multi-paradigme. Son système de type permet un développement logiciel d'une grande fiabilité. De plus en plus d'entreprises, par exemple Facebook et JaneStreet, l'intègrent dans leurs projets. D'autre part, notre langage cible sera PHP, un langage multi-paradigme et le plus utilisé dans le monde du web, son typage dynamique permet une grande flexibilité de programmation. Les langages typés dynamiquement permettent un développement rapide, mais sacrifient la capacité à détecter des erreurs tôt. Il existe certaines solutions qui permettent d'introduire le typage statique à PHP, comme Hack de Facebook. Dans ce travail, nous avons choisi plutôt de compiler un programme OCaml vers du code PHP lisible ayant la même sémantique, qui pourrait profiter du typage statique d'OCaml et facile à maintenir par des développeurs non familiers avec OCaml.

Dans un premier temps, nous allons définir le contexte en présentant entre autres un outil similaire tel que *js_of_ocaml*, ensuite on passera à la conception et nous détaillerons le choix de l'approche AST typé vers PHP et décrierons les squelettes de traduction de certaines expressions OCaml et l'équivalent en PHP et enfin nous conclurons par l'expérimentation de *php_of_OCaml*.

Chapitre 1

Contexte

Durant ce chapitre, nous allons présenter les différentes phases de compilation d'un programme OCaml et nous détaillerons chaque étape de compilation. Nous présenterons également *js_ of_ ocaml*, un compilateur qui prend en entrée du code OCaml et qui génère du JavaScript.

1.1 Compilation en OCaml

Les phases de génération de code du compilateur Objective CAML sont détaillées à la figure 1.1.

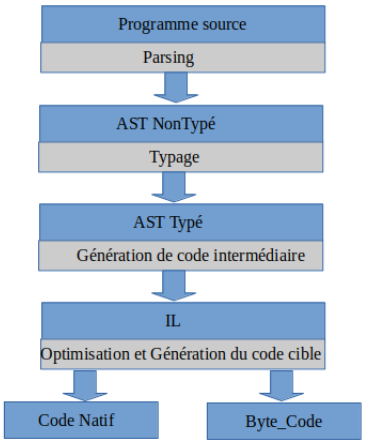

FIGURE 1.1 – Génération de code natif et bytecode.

La première phase de compilation est l'analyse lexicale et consiste à lire le flot de caractère qui constitue le programme source et les regrouper en séquences de caractères significatifs appelés lexèmes. Durant la deuxième partie, l'analyseur syntaxique se sert du premier composant des unités lexicales produites par l'analyseur, à ce niveau le parser se charge de vérifier la syntaxe et la construction grammaticale du programme vis-à-vis de la spécification du langage afin de construire une représentation intermédiaire

arborescente appelée AST [1]. Ensuite à la phase d'analyse sémantique on prend l'arbre abstrait et les informations de la table des symboles pour vérifier que le programme source est sémantiquement correcte, cela en collectant des informations sur les types, cette vérification de type s'effectue statiquement. Finalement à la sortie de cette phase nous aurons un AST typé.

1.1.1 AST et typage

Un arbre de syntaxe abstraite est un arbre dont la structure ne garde plus trace des détails de l'analyse, mais seulement de la structure du programme. Dans un arbre de syntaxe abstraite, construit à partir d'un arbre syntaxique, on n'a pas besoin de garder trace des terminaux qui explicitent le parenthésage, vu qu'on le connaît déjà grâce à sa structure arborescente. De même, tous les terminaux de la syntaxe concrète (comme les virgules, les points virgules, les guillemets, les mots clefs etc.) qui ont pour but de signaler des constructions particulières, n'ont plus besoin d'apparaître.

À la fin de la phase de l'analyse sémantique on aura un arbre de syntaxe abstraite typé (AST typé) portant les informations de type sur les expressions.

1.1.2 Langage intermédiaire

La représentation interne du code généré par le compilateur est appelée langage intermédiaire (IL).
Le but du code intermédiaire est de passer d'une structure arborescente à une (bonne) structure linéaire et de préparer la sélection d'instructions. Les détails dépendant de l'architecture sont relégués à une phase ultérieure de sélection d'instructions.

La phase d'optimisation du code tente d'améliorer le code intermédiaire, de sorte que le code cible soit plus léger, court et consommant peu d'énergie.

1.1.3 Génération du code

Il existe deux sorties possibles du compilateur OCaml, l'un compilant vers du code-octet (byte-code), l'autre vers du code natif.

1. Compilation vers du code-octet

   Le code-octet peut être exécuté par une machine virtuelle OCaml sur d'autres architectures que celle pour laquelle il a été compilé. Des performances moindres sont la contrepartie de cette portabilité. Par ailleurs, dans le cas d'utilisation de bibliothèques C, ces bibliothèques doivent être présentes sur la machine d'exécution.

2. Compilation vers du code natif

   La compilation vers du code natif permet d'obtenir de bien meilleures performances pour l'exécutable produit. Cependant, l'exécutable obtenu ne peut pas être exécuté sur d'autres architectures que celle sur laquelle il a été compilé.

1. Abstract Syntax Tree ou Arbre de Syntaxe Abstraite.

1.2 *js_ of_ ocaml*

Nous présentons les grandes lignes du compilateur *js_ of_ ocaml* qui part du bytecode OCaml et génère un programme cible en JavaScript. Ce compilateur traduit le bytecode en une représentation statique à l'affectation unique sur laquelle les optimisations sont effectuées, avant de générer un programme JavaScript sémantiquement équivalent.

Jérôme Vouillon et Vincent Balat, dans leur projet *js_ of_ ocaml* Compiler [2], partent du bytecode pour générer du code JavaScript. Ceci est un choix judicieux car nous profitons des optimisations, la machine abstraite de Zinc (ZAM) offre une API stable et la traduction est directe. Il n'est pas nécessaire de modifier le compilateur à chaque nouvelle version du langage. De plus, ils ont conçu un moyen pour utiliser facilement les API de JavaScript disponibles, bien qu'elles soient orientées objet et que la convention d'appel de JavaScript diffère de celui d'OCaml. Leur approche leur permet également de profiter du typage statique d'OCaml pour avoir moins de bugs au runtime.
Parmi les atouts de *js_ of_ ocaml*, le code généré n'a aucune dépendance particulière avec OCaml, il peut tourner directement dans un navigateur.

Un tel compilateur est également pratique à utiliser, il peut simplement être ajouté à une installation existante des outils de développement. Les bibliothèques déjà compilées peuvent être utilisées directement, sans avoir à réinstaller quoi que ce soit.

2. http ://ocsigen.org/js_of_ocaml/

Chapitre 2

Conception

Ce chapitre portera sur la phase de conception, en mettant en place tous les mécanismes qui nous permettront la génération du code PHP à partir de l'AST typé d'un programme OCaml. En premier lieu, nous allons présenter quelques traits du langage que notre outil est capable de traduire [1] et des squelettes de traduction de certaines expressions du langage.
On rappelle que PHP est un langage de programmation interprété, typé dynamiquement, les erreurs de programmation ne seront détectées qu'à l'exécution.

2.1 Compilation avec *php_ of_ OCaml*

La figure 2.1 présente les étapes de compilation de notre outil *php_ of_ OCaml* en utilisant le compilateur OCaml, dont la sortie sera un fichier PHP sémantiquement équivalent au programme source OCaml.

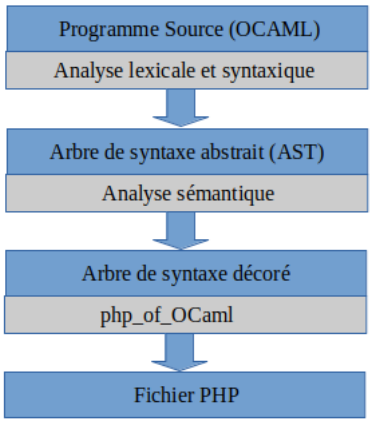

FIGURE 2.1 – Les étapes de compilation avec *php_ of_ OCaml*.

Le typage statique d'OCaml permet d'avoir moins de bugs au runtime, de détecter certaines erreurs de programmation.

1. Disponible à l'adresse : https ://github.com/GHERSASofiane/php_of_OCaml.git

En partant d'un programme OCaml typé statiquement pour générer un code sémantiquement équivalent en PHP qui sera exempt de certaines erreurs de programmation par construction.

2.2 Choix de l'approche AST Typé vers PHP

Contrairement à *js_ of_ ocaml*, nous avons choisi l'AST typé produit après la compilation d'un programme OCaml car il est simple à manipuler et nous pourrons profiter des informations de types.

On part de la chaîne de compilation du compilateur OCaml et on s'arrête après la phase de typage pour s'assurer que notre programme est bien typé. Ensuite notre outil prend comme entrée cet AST bien typé et génère un programme PHP équivalent sémantiquement en faisant en sorte de produire une sortie lisible pour des programmeurs PHP.

php_ of_ OCaml parcourt l'AST typé récursivement et applique des fonctions de traduction sur chacun de ses noeuds, le tout en maintenant un contexte permettant de traiter les cas spéciaux (par exemple : les return ou les affectations).

2.3 Grammaire du langage source

Nous avons opté pour la traduction de certains traits du langage OCaml. Nous présentons ci-dessous la grammaire du sous-ensemble du langage que nous allons traduire :

```
EXPR ::= IDENT
    | EXPR ';' EXPR
    | 'if' EXPR 'then' EXPR 'else' EXPR (optional)
    | 'for' IDENT '=' EXPR 'to' EXPR 'do' ( EXPR )* 'done'
    | 'while' EXPR 'do' (EXPR)+ 'done'
    | '{' (IDENT '=' EXPR) ( ';' IDENT '=' EXPR)* '}'
    | 'match' EXPR 'with' ( ('|' PATTERN '->' EXPR)+)
    | 'try' EXPR 'with' '|' IDENT '->' EXPR
    |  NUMS
    | 'let' 'rec'? IDENT (IDENT)* '=' EXPR
    | '[' (EXPR ';')* ']'
    | '(' (EXPR) (','EXPR)+ ')'
    | '[|' (EXPR ';')* '|]'
PATTERN ::=
      PATT_LIST
    | PATT_ARRAY
    | PATT_TUPLE
    | IDENT
    | '_'
```

```
PATT_LIST ::=
            | IDENT
            | IDENT ':' ':' PATT_LIST
            | '[' ']'
```

2.4 Squelettes de traduction

Nous définissons la traduction à travers un ensemble de règles que nous détaillons ci-dessous.

Notre fonction de traduction sera $[\![.]\!] \in (\texttt{EXPR} \rightarrow \texttt{EXPR}_{PHP})$.

— Boucles FOR

$$\frac{[\![for \quad IDENT \quad = \quad EXPR \quad to \quad EXPR \quad do \quad (EXPR) \quad done]\!]}{for(\$\,[\![IDENT]\!] = [\![EXPR]\!]\,;\,\$\,[\![IDENT]\!] <= EXPR\,;\,\$\,[\![IDENT]\!] + +)\{[\![< (EXPR) >]\!]\}}$$

— Boucles WHILE

$$\frac{[\![while \quad COND \quad do \quad (EXPR) * \quad done]\!]}{WHILE \quad ([\![COND]\!])do \quad \{[\![EXPR]\!]\}}\ {}^{2}$$

— Tableaux

$$\frac{[\![let \quad IDENT \quad = \quad [|(EXPR)? \quad (;(EXPR)) * |] \quad]\!]}{\$\,[\![IDENT]\!] \quad = array \quad (\quad ([\![EXPR]\!])? \quad (,([\![EXPR]\!]))* \quad)}$$

— N-uplets

$$\frac{[\![let \quad IDENT \quad = \quad ((EXPR)? \quad (,(EXPR))*)]\!]}{\$\,[\![IDENT]\!] \quad = array \quad (([\![EXPR]\!])? \quad (,([\![EXPR]\!]))* \quad)}$$

— Exceptions

$$\frac{[\![try \quad EXPR \quad |withIDENT \quad -> \quad EXPR]\!]}{try\{ \quad [\![EXPR]\!] \quad \}catch(Exception[\![IDENT]\!]) \quad \{ \quad [\![EXPR]\!] \quad \}}$$

2.4.1 Déclaration de variables

Nous illustrons ci-dessous les cas de traduction des déclarations de variables :

1. Déclaration globale : let ident = expression ; ;

   $$\frac{[\![let \quad IDENT \quad = \quad EXPR]\!]}{\$[\![IDENT]\!] = \quad [\![EXPR]\!]}$$

   Le tableau ci-dessous illustre un exemple de ce cas.

Ocaml	PHP
let x =10 ; ;	**$x** = (10);

2. Déclaration globale et/ou locale : let ... and ident2 = expression2 and ident3 = ...

   let IDENT = let IDENT = <EXPR> in let IDENT = <EXPR> in ...

2. À ce niveau, nous considérons COND comme étant une expression booléenne.

Dans ce cas, on aura une séquence de **déclarations locales** qui seront traduites au premier lieu, on va suspendre **la déclaration globale**, la variable avec le contexte le plus général, ensuite nous allons vérifier si le nom de la variable avec le contaxte le plus général existe dans la liste des déclarations locales, dans ce cas on renomme cette variable globale et on lui affecte la dernière instruction, l'exemple ci-dessous illustre ce cas particulier.

OCaml	PHP
let x = let x=5 in	$\$x = 5;$
let a=(7*2) in	$\$a = (7 * 2);$
let y=1 in	$\$y = 1;$
y+a+x ; ;	**$x1**=(($x+$a)+$y) ;

2.4.2 Fonctions

Pour la traduction des fonctions OCaml, on aura deux cas. Les fonctions récursives, le nom de la fonction est précédé par un mot clé **rec** et les fonctions non récursives sans le mot clé **rec**.

Par contre en PHP il n'y a pas de différence concernant la déclaration des fonctions.

$$\frac{[\![let \quad rec? \quad IDENT \quad (IDENT)+ \quad = \quad EXPR]\!]}{function \quad [\![IDENT]\!] \quad (\quad [\![IDENT]\!]? \quad (',' \ [\![IDENT]\!])*) \quad \{ \quad [\![EXPR]\!] \quad \}}$$

2.4.3 Pattern Matching

```
match EXPR with
  |p1 -> EXPR1
  :
  |pn -> EXPRn
```

Nous distinguons deux cas concernant le pattern matching, selon le pattern **match PATTERN with**.

1. Le type du pattern est un type primitif (int, bool, char, string ..)
   dans ce cas nous allons utiliser le **switch** de PHP pour pouvoir comparer la même variable (ou expression de type primitive) avec des valeurs différentes, et d'exécuter différentes parties de code suivant la valeur à laquelle elle est égale.

Ocaml	PHP

let match b with
| 'A' → print_string "vowel"
| 'E' → print_string "vowel"
| 'B' → print_string "consonant"
| _ → print_string "other"
	switch ($b) {
	case 'A' : echo ("vowel") ; break ;
	case 'E' : echo ("vowel") ; break ;
	case 'B' : echo ("consonant") ; break ;
	default : echo ("other") ; break ;
	}

2. L'expression est de type List, dans ce cas nous allons regarder la taille de chaque proposition **p** de | p → EXPR, pour pouvoir utiliser le même mécanisme des **if..then..else..** imbriqués.

Ocaml	PHP

let match x with
| [] → print_int 7
| a : :b : :[] → print_int 19
| a : :b : :c → print_int 39
| _ → print_int 98
	
	if (sizeof($x) == 0) { $hd=$x[0] ; $rst=array_slice($x,1) ;
	echo (7) ; }else{
	if (sizeof($x) == 2) { $hd=$x[0] ; $rst=array_slice($x,1) ;
	echo (19) ;}else{
	if (sizeof($x) >= 2) { $hd=$x[0] ; $rst=array_slice($x,1) ;
	echo (39) ;}else{
	echo (98) ; } } }

2.4.4 Enregistrements

$$\frac{[\![\{(IDENT \quad = EXPR(;(IDENT \quad = EXPR))*\}]\!]}{[\quad ([\![IDENT]\!]' =>' [\![EXPR]\!] \quad (,([\![IDENT]\!]' =>' [\![EXPR]\!]))*]}$$

Voici un exemple de traduction des enregistrements OCaml en des tableaux associatifs PHP :

Ocaml	PHP
let user = {	$user = [
id = 2 ;	'id' => 2,
pseudo = "breakLIMITS" ;	'pseudo' => "breakLIMITS",
age = 23	'age' => 23
}	] ;

2.4.5 Return

Nous avons besoin de sauvegarder le contexte de certaines expressions afin de pouvoir effectuer la traduction des return en OCaml, sachant que cette sauvegarde s'effectue lorsqu'on essaye de traduire une fonction. Nous illustrons ça avec l'exemple ci-dessous.

```
let split chaine separateur =
  let result = ref [] in
  begin
        let ch = ref "" in
        for i = 0 to String.length chaine - 1 do
              begin
                 if chaine.[i] = separateur then
                 begin
                     result := !ch :: !result; ch := ""
                 end
                 else
                     ch := !ch^(String.make 1 chaine.[i])
              end
        done;
        result := !ch :: !result;
  end;
 !result;;
```

Avant de quitter le corps de la fonction le compilateur teste si la dernière séquence du corps de la fonction s'agit d'un type de retour. Dans ce cas, le compilateur php_of_OCaml applique la fonction de traduction sur ce noeud. Dans cet exemple nous avons comme type de retour la variable **result**.

```
function split($chaine,$separateur){
            $result = (array());
          $ch = ("");
         for ($i=0; $i <= (strlen($chaine)-1); $i++)
          {
             if ((($chaine[$i]) == $separateur)) {
                         array_unshift(($result),($ch));
                         ($ch = "" );
                    }else{
                     ($ch = (($ch).(($chaine[$i]))));
                    }
          }
        array_unshift (($result),($ch));
 return ($result);
 }
```

2.4.6 Primitives et fonctions de la bibliothèque standard

Nous avons stocké un sous-ensemble des primitives et fonctions de la bibliothèque standard OCaml dans une table d'association avec leurs équivalent sémantique en PHP. Chaque appel à une primitive engendre

une recherche dans la table pour vérifier si la primitive a bien été définie afin d'extraire de la table sa correspondante en PHP. Nous présentons ci-dessous quelques unes de ces traductions.

Ocaml	PHP
List.length lst	sizeof($lst)
List.nth lst i	$lst[i]
List.append l1 l2	array_merge($l1, $l2)
String.length str	strlen($str)
String.get str i	$str[i]
string_of_int	strval($str)
"Bon"^"jour"	"Bon"."jour"
print_string "Bonjour"	echo "Bonjour"
print_endline "Bonjour"	echo "Bonjour"." n"
open_in "Inscription.csv"	fopen ("Inscription.csv" , "r")
open_out "Inscription.csv"	fopen ("Inscription.csv" , "w")
close_in "Inscription.csv"	fclose ("Inscription.csv")
result := "Bonjour" : :!result	array_unshift ((($result)) , (("Bonjour")))

Chapitre 3

Expérimentation

Pour tester notre compilateur *php_ of_ OCaml*, nous avons développé une application de gestion d'utilisateurs en OCaml, nous allons tester le fonctionnement du code PHP sur un serveur web Apache [1].

3.1 Application web d'enregistrement d'utilisateurs

Nous manipulons un fichier CSV, qui sera notre base de données, contenant l'ensemble des informations de nos utilisateurs. Nous détaillerons notre étude ci-dessous.

1. Premièrement, nous partons d'un fichier source OCaml, contenant les fonctions qui permettent d'ajouter, modifier, supprimer, récupérer les informations d'un utilisateur donné et également la récupération de l'ensemble des utilisateurs.
2. Nous compilons ce programme OCaml avec notre outil *php_ of_ OCaml* et nous obtiendrons un fichier PHP en sortie, un équivalent sémantique des fonctions déclarées dans le fichier OCaml source.
3. La dernière phase consiste à récupérer le fichier généré et le mettre à la racine de notre application afin d'invoquer l'ensemble de ses fonctions. Sachant que les signatures de ses fonctions sont les même que celles que nous avons défini dans le fichier source OCaml.

Nous avons mis en annexe l'extrait de quelques fonctions écrites en OCaml dans le fichier CSV.ml et l'équivalent sémantique en PHP dans le fichier CSV.php.

1. Apache est un serveur web permettant de partager des pages web stockées localement.

Nous schématisons les entités qui participent au fonctionnement de notre application web.

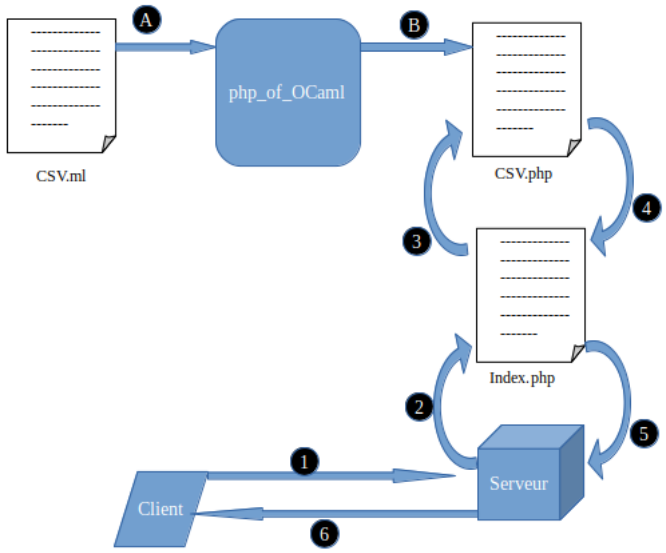

FIGURE 3.1 – Mise en oeuvre de *php_ of_ OCaml*.

A : L'utilisation de *php_ of_ OCaml* pour la compilation.

B : Récupération du fichier généré avec l'outil et l'intégrer dans notre application.

1 : Requête envoyée par le client.

2 : Demande la génération de la réponse à la demande du client.

3 : Invocation des fonctions obtenues après la compilation.

4 : Récupération des données retournées par les fonctions invoquées.

5 : Réception de la réponse à retourner au client.

6 : Réponse à la requête.

Chapitre 4

Conclusion

Dans le cadre de notre PSTL [1] nous avons pu développer un compilateur *php_ of_ OCaml* qui permet à partir d'un programme OCaml de générer du PHP exécutable sur un serveur web. Ainsi, avec la vérification statique des types assurée par le compilateur OCaml, nous obtenons un programme PHP lisible et qui est assuré, par construction, de ne pas avoir d'erreurs au runtime.

Nous avons pu mettre en corrélation les aspects théoriques et pratiques, sans éclipser les difficultés que nous avons rencontrées dans ce projet, c'est-à-dire la familiarisation avec le langage fonctionnel et l'extractions des informations concernant les expressions récupérées dans l'AST typé produit par le compilateur OCaml qui s'avère abstrait au début de notre étude.

Au terme de ce projet, notre sentiment est très positif ; nous avons effectivement acquis de nombreuses connaissances profondes de compilation ainsi que le fonctionnement des compilateurs en général et le compilateur OCaml en particulier. Nous avons acquis des compétences nouvelles également la programmation fonctionnelle d'une part en OCaml et un langage multi-paradigme typé dynamiquement d'autre part, nous avons pu développer l'esprit d'équipe et la gestion de projets grâce à GitHub, le service web d'hébergement et de gestion de développement de logiciels. Pour la rédaction du rapport nous avons appris et utilisé le langage de composition de documents LaTeX.

Comme perspectives, nous allons implémenter notre compilateur en ligne pour que l'on puisse effectuer nos traductions sur un serveur web, nous allons traduire les notions d'objets OCaml sachant que la programmation par objets a été intégrée au langage PHP dans sa version 4 et également nous allons étendre le langage supporté afin de traiter le système de modules OCaml.

Notre voeu en clôturant ce projet est que les ressources que nous avons pu développées soient utiles et utilisées par un maximum d'étudiants et de professionnels en informatique.

1. Projet Science Technologie du Logiciel

Bibliographie

[1] Jeffrey D.Ullman Alfred V.Aho, Ravi Sethi. *Compilers principles, Techniques, and tools.* Pearson Education, 1986.

[2] Bruno Pagano Emmanuel Chailloux, Pascal Manoury. *Développement d'applications avec Objective Caml.* ÉDITIONS O'REILLY, PARIS, 2000.

[3] INRIA. Un makefile générique pour objective caml. [online]. available :. *https ://caml.inria.fr/pub/old_ caml_ site/FAQ/Makefile_ ocaml-fra.html*, [Accessed : 12-Jan-2017].

[4] Benoit Vaugon Michel Mauny. Ocamlcc - traduire ocaml en c en passant par le bytecode. damien pous and christine tasson. *JFLA - Journées francophones des langages applicatifs*, Feb 2013, Aussois, France 2013.

[5] Ocsigen. js_of_ocaml. *http ://ocsigen.org/js_ of_ ocaml/*, [Accessed : 12-Jan-2017].

[6] Karl Berry Paul W. Abrahams, Kathryn A. Hargreaves. *TEX for the Impatient.* Addison-Wesley, 1990.

[7] Jérôme Vouillon Vincent Balat. From bytecode to javascript, the js_of_ocaml compiler. [online]. available :. *https ://www.irif.fr/ balat/publications/vouillon_ balat-js_ of_ ocaml.pdf*, [Accessed : 18-Jan-2017].

Annexe A

CSV.ml

```
1  open Printf
2  type user = {
3            mutable id: int;
4            mutable nom: string;
5            mutable date_naissance: string;
6            mutable mail: string;
7            mutable telephone: string;
8             }
9
10
11 let split chaine separateur =
12   let result = ref [] in
13   begin
14         let ch = ref "" in
15         for i = 0 to String.length chaine - 1 do
16               begin
17                  if chaine.[i] = separateur then
18                  begin
19                      result := !ch :: !result; ch := ""
20                  end
21                  else
22                      ch := !ch^(String.make 1 chaine.[i])
23               end
24         done;
25         result := !ch :: !result;
26   end;
27  !result
28
29 let lire_file file =
```

```
30   let entree = (open_in file)
31   and result = ref [] in
32   begin
33        try
34            let ligne = ref (split (input_line entree) ';' ) in
35            let nb_champs = (List.length !ligne) in
36            while true do
37                     if (List.length !ligne) != nb_champs then
38                         print_endline "Erreur de lecture"
39                     else
40                         result := List.append !result [ !ligne ];
41                         ligne := (split (input_line entree) ';' )
42                     done
43        with
44          | End_of_file -> close_in entree
45          (* fclose ( $file ); *)
46   end;
47
48   !result
49
50
51 let write usr =
52   let id = (string_of_int (usr.id)) in
53   let file = "Inscription.csv" in
54    let x = id^";"^usr.nom^";"^usr.date_naissance^";"^usr.mail^";"^usr.telephone^"\n" in
55     let flag_list = [Open_append] in
56       let sortie = open_out_gen flag_list 755 file in
57          output_string sortie x;
58          close_out sortie
59
60
61  let clean file =
62   let out_chanel = open_out file in
63       output_string out_chanel "";
64       close_out out_chanel
65
66
67   let delete id =
68   let file = "Inscription.csv" in
69   let id = string_of_int id in
70     let list_ligne = lire_file file in
71     let l = ref list_ligne in
```

```
72      clean file;
73        for i = 0 to List.length !l -1 do
74        begin
75             let subList = (List.nth !l i) in
76             if((List.nth subList 4) <> id) then
77                 begin
78                   let usr_search = ref {
79                                id = int_of_string (List.nth subList 4) ;
80                                nom = (List.nth subList 3);
81                                date_naissance = (List.nth subList 2);
82                                mail = (List.nth subList 1);
83                                telephone = (List.nth subList 0);
84                                } in
85                  write (!usr_search)
86                 end
87           else ()
88
89        end
90      done
91
92  let search id_usr =
93       let usr_search = ref {
94                           id = -1 ;
95                           nom = "";
96                           date_naissance = "";
97                           mail = "";
98                           telephone = "";
99                           } in
100     let id = (string_of_int (id_usr)) in
101     let file = "Inscription.csv" in
102     let list_ligne = lire_file file in
103     let l = ref list_ligne in
104       for i = 0 to List.length !l -1 do
105       begin
106         let subList=(List.nth !l i) in
107         if((List.nth subList 4) = id) then
108         begin
109             !usr_search.id <- id_usr;
110             !usr_search.nom <- (List.nth subList 3);
111             !usr_search.date_naissance <- (List.nth subList 2);
112             !usr_search.mail <- (List.nth subList 1);
113             !usr_search.telephone <- (List.nth subList 0)
```

```
114
115         end
116       end
117
118       done;
119   !usr_search
120
121
122 let get_list () =
123   let list_users = ref [] in
124   let file = "Inscription.csv" in
125   let list_ligne = lire_file file in
126   let l = ref list_ligne in
127   for i = 0 to List.length !l -1 do
128   begin
129     let usr_search = ref {
130                               id = int_of_string (List.nth (List.nth !l i) 4) ;
131                               nom = (List.nth (List.nth !l i) 3);
132                               date_naissance = (List.nth (List.nth !l i) 2);
133                               mail = (List.nth (List.nth !l i) 1);
134                               telephone = (List.nth (List.nth !l i) 0);
135                               } in
136     list_users := usr_search :: !list_users ;
137   end
138   done;
139   !list_users
```

Annexe B

CSV.php

```php
<?php



function split ( $chaine, $separateur ){
    $result =(array());
  $ch =("");
 for ($i=0; $i <= (strlen($chaine)-1); $i++)
  {
   if ( ( ($chaine[$i]) == $separateur) ) {
       array_unshift ( ($result) , ($ch) ) ;
       ( $ch = "" );
   }else{
     ( $ch = (($ch). ( ($chaine[$i]) )) );

   }
  }
array_unshift ( ($result) , ($ch) ) ;

  return ($result) ;
}


function lire_file ( $file ){
    $entree =fopen($file , "r" );
  $result =(array());

  try{
  $ligne =(split(fgets($entree),';'));
```

```
30    $nb_champs =sizeof(($ligne));
31   while (true )
32    {
33       if ( (sizeof(($ligne))!=$nb_champs) ) {
34  break;
35
36  }
37   else{
38    array_merge(($result), ( ($ligne) ) );
39
40  }
41  ( $ligne = split(fgets($entree),';') );
42
43    }
44
45    }catch (Exception $e){
46
47 fclose($entree);
48       }
49
50    return ($result) ;
51  }
52
53 function write ( $usr ){
54     $id =strval($usr['id']);
55   $file = "Inscription.csv";
56   $x = ($id.(";".($usr['nom'].(";".($usr['date_naissance'].(";".($usr['mail'].(";".($usr['
       telephone'])."\n") ) ) ) ) ) ) ) ) ;
57     file_put_contents($file, $x, FILE_APPEND | LOCK_EX);
58     fclose($sortie);
59  }
60
61 function clean ( $file ){
62      $out_chanel =fopen($file , "w" );
63      fputs($out_chanel,"");
64
65     fclose($out_chanel);
66  }
67
68
69 function delete ( $id ){
70     $file = "Inscription.csv";
```

```
71    $id =strval($id);
72    $list_ligne = lire_file ($file);
73    $l =($list_ligne);
74   clean ($file);
75   for ($i=0; $i <= (sizeof(($l))-1); $i++)
76    {
77       $subList = ($l)[$i];
78   if ( ($subList[4]<>$id) ) {
79    $usr_search =
80   [
81      'id' => intval($subList[4]),
82      'nom' => $subList[3],
83      'date_naissance' => $subList[2],
84      'mail' => $subList[1],
85      'telephone' => $subList[0]
86     ] ;
87   write (($usr_search));
88
89  } else{
90
91  }
92    }
93
94  }
95
96
97 function search ( $id_usr ){
98      $usr_search =
99   [
100     'id' => -1,
101     'nom' => "",
102     'date_naissance' => "",
103     'mail' => "",
104     'telephone' => ""
105    ] ;
106   $id =strval($id_usr);
107  $file = "Inscription.csv";
108   $list_ligne = lire_file ($file);
109   $l =($list_ligne);
110  for ($i=0; $i <= (sizeof(($l))-1); $i++)
111   {
112      $subList = ($l)[$i];
```

```
113   if ( ($subList[4] == $id) ) {
114
115  ($usr_search)['id']=$id_usr;
116
117  ($usr_search)['date_naissance']=$subList[2];
118
119  ($usr_search)['telephone']=$subList[0];
120
121  }
122
123    }
124    return ($usr_search) ;
125  }
126
127
128
129 function get_list ( ){
130      $list_users =(array());
131   $file = "Inscription.csv";
132    $list_ligne = lire_file ($file);
133    $l =($list_ligne);
134   for ($i=0; $i <= (sizeof(($l))-1); $i++)
135    {
136        $usr_search =
137   [
138      'id' => intval(($l)[$i][4]),
139      'nom' => ($l)[$i][3],
140      'date_naissance' => ($l)[$i][2],
141      'mail' => ($l)[$i][1],
142      'telephone' => ($l)[$i][0]
143     ] ;
144  array_unshift ( ($list_users) , $usr_search ) ;
145    }
146
147    return ($list_users) ;
148  }
149
150
151
152 ?>
```