Université Gustave Eiffel

Examen final du 6 juillet 2021

Compilation

—Licence d'informatique - 3e année—

Durée : 2 h. Les quatre parties sont indépendantes. Documents papier autorisés. Appareils électroniques interdits. Le barème donné ci-dessous est indicatif.

▸ **Exercice 1.** ***Accès mémoire en* `nasm` *64 bits orienté little-endian (4 points)***

Après exécution du code ci-dessous, que contiennent les registres `rax`, `rbx`, `rcx` *et* `rdx` *?*

```
section .data
my_bytes: db 1, 2, 3, 4
my_words: dd 4, 3, 2, 1
section .text
mov rax,0
mov al,byte[my_bytes]
mov rbx,0
mov bl,byte[my_bytes+3]
mov rcx,0
mov cx,word[my_words+2]
mov rdx,0
mov dl,byte[my_words+3]
```

▸ **Exercice 2.** ***Lecture de code en* `nasm` *64 bits : pile (3 points)***

Après exécution du code ci-dessous (la totalité des 26 instructions), que contiennent les registres `rax` *et* `rbx` *?*

```
push 2
push 3
push 3
pop rbx
pop rax
imul rax, rbx
push rax
push 2
push 3
```

```
pop rbx
pop rax
imul rax, rbx
push rax
pop rbx
pop rax
sub rax, rbx
push rax
push 1
```

```
pop rbx
pop rax
add rax, rbx
push rax
pop rbx
pop rax
imul rax, rbx
push rax
```

▸ **Exercice 3.** ***Expressions de type en C (5 points)***

On représente les expressions de type en C à l'aide des types simples (`int`, `float`*...), de* `void` *et de quatre constructeurs de type :*

- `pointeur`(t), *pointeur sur* t *;*
- `tableau`$(x..y,\ t)$, *tableau de* t *avec des indices variant de* x *à* y *;*
- `fonction`(p,t), *fonction qui prend des paramètres de type* p *et renvoie un* t*.*
- $p \times q$, *pour combiner deux types* p *et* q*.*

Pour chacune des déclarations suivantes (`c`, `d`, `e`, `f` *et* `g`*),*

- *dessinez un arbre du déclarateur pour préciser toutes les priorités ;*

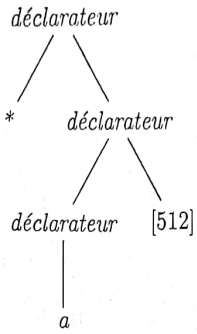
— *donnez l'expression de type de la variable déclarée en écrivant une formule.*

Exemple :

```
int *a[512]; int *b();
```

— *Pour* `a` *:*

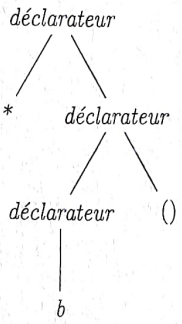
— *arbre du déclarateur :*



— *expression de type :* `tableau(0..511,pointeur(int))`

— *Pour* `b` *:*

— *arbre du déclarateur :*



— *expression de type :* `fonction(void,pointeur(int))`

C'est à vous : `float **c, d; long (*e)(), (*(*f)())(), (*g[8])();`

▸**Exercice 4.** ***Développement en Bison (8 points)***

Le but de cet exercice est de faire un programme `bison` *qui crée des arbres abstraits. On donne un module en C pour manipuler des arbres,* `abstract-tree.c` *et* `abstract-tree.h`*.*

```
/* abstract-tree.h */
typedef enum {
  DeclVars,
  Declarateurs,
  Type,
  Ident,
  Comma,        /* ',' */
  Semicolon,    /* ';' */
  Empty,        /* epsilon */
  DeclVar
} Kind;
typedef struct Node {
```

```
  Kind kind;
  struct Node *firstChild, *nextSibling;
  int lineno;
} Node;
Node *makeNode(Kind kind);
void addSibling(Node *node, Node *sibling);
void addChild(Node *parent, Node *child);
#define FIRSTCHILD(node) node->firstChild
#define SECONDCHILD(node) node->firstChild->nextSibling
#define THIRDCHILD(node) node->firstChild->nextSibling->nextSibling
```

```
/* abstract-tree.c */
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "abstract-tree.h"
extern int lineno;       /* from lexer */
Node *makeNode(Kind kind) {
  Node *node = malloc(sizeof(Node));
  if (!node) {
    printf("Run out of memory\n");
    exit(1);
  }
  node->kind = kind;
  node->firstChild = node->nextSibling = NULL;
  node->lineno=lineno;
  return node;
}
void addSibling(Node *node, Node *sibling) {
  Node *curr = node;
  while (curr->nextSibling != NULL) {
    curr = curr->nextSibling;
  }
  curr->nextSibling = sibling;
}
void addChild(Node *parent, Node *child) {
  if (parent->firstChild == NULL) {
    parent->firstChild = child;
  }
  else {
    addSibling(parent->firstChild, child);
  }
}
```

On donne une grammaire des déclarations de variables au format `bison` *:*

DeclVars : *DeclVars TYPE Declarateurs ';'*
| ;

Declarateurs : *Declarateurs ',' IDENT*
| *IDENT*
;

TYPE reconnait `float`, `long` *et* `int`. *IDENT reconnait les autres identificateurs.*

1. *Dessinez l'arbre de dérivation de :*

   ```
   float pi, alpha; long p, a;
   ```

   Rappel : l'arbre de dérivation est celui dont tous les noeuds internes sont occupés par les non-terminaux de la grammaire.

2. *Écrivez des actions* `bison` *qui construisent en mémoire un arbre exactement semblable à l'arbre de dérivation. On suppose qu'on a déjà un programme* `flex` *qui donne des attributs aux lexèmes TYPE, IDENT, ',' et ';'. Ces attributs sont des pointeurs vers des* `Node` *déjà alloués, dont le champ* `kind` *vaut respectivement* `Type`, `Ident`, `Comma` *et* `Semicolon`. *On ne demande pas d'écrire le programme* `flex`.

3. *Écrivez un nouvel ensemble d'actions* `bison` *qui construisent en mémoire un arbre abstrait conforme à la figure 1.*

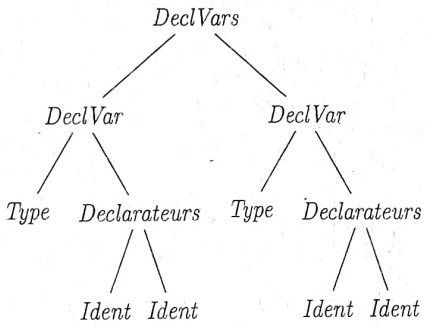


FIGURE 1 – Arbre abstrait de `float pi, alpha; long p, a;`

4. *Écrivez un dernier ensemble d'actions* `bison` *qui construisent en mémoire un arbre abstrait conforme à la figure 2. On ne demande pas d'indiquer dans l'arbre abstrait la valeur des types, ni la chaine de caractères de l'identificateur, on veut seulement la structure des arbres.*

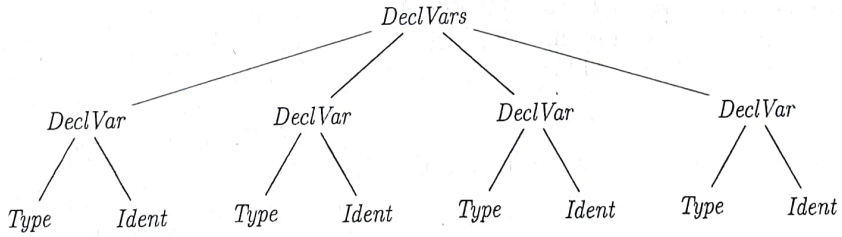


FIGURE 2 – Arbre abstrait de `float pi, alpha; long p, a;`