**DÉPARTEMENT D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE**

**COURS:** IFT 2030 (A2002) – Concepts des langages de programmation

**NOM DU PROFESSEUR:** Philippe Langlais

**EXAMEN FINAL**

Date : mardi 10 décembre 2002
Heure : 13h30 - 16h30
Salle : B-2285 - M-C L-G 3200 JB

| |
|---|
| Code permanent: |
| Nom: |
| Prénom: |

**DIRECTIVES PÉDAGOGIQUES:**

- Inscrivez votre nom et votre code permanent dès maintenant dans l'encadré.
- Aucune documentation autorisée (hors dictionnaire français/L où L est votre langue maternelle).
- Répondez à même le sujet. Si vous manquez de place (ce qui serait étonnant) utilisez le verso des feuilles en l'indiquant clairement.
- Le barème est donné à titre indicatif et peut-être modifié lors de la correction.
- Soyez bref et précis.
- Lorsqu'un langage vous est imposé (ex: C), si vous répondez à l'aide d'un autre langage (ex: Java), votre réponse ne sera pas corrigée (encore moins si vous inventez une syntaxe hybride (ex: CAVA).
- Il y a dans le sujet des questions qui sont composées de plusieurs sous-questions (ex: expliquez puis donner un exemple). Ne répondre qu'à une partie de la question est pénalisant.
- Je note ce qui est sur la copie et pas ce que vous avez dans la tête au moment où vous composez.
- Prenez le temps de lire une fois le sujet avant de commencer à répondre; bonne chance.
- Le total des points à prendre est de 105. La note maximale que vous pouvez cependant obtenir à cet examen est 100.

| | | | |
|---|---|---|---|
| Question 1: | /25 | Question 4: | /20 |
| Question 2: | /20 | Question 5: | /25 |
| Question 3: | /10 | | |
| Total: | | /100 | |

**Question A:** *Le coin: Cours* (25 points)

**A-1** Expliquez la différence entre portée lexicale et dynamique. Donnez un exemple de code (dans un pseudo langage) qui illustre la différence entre les deux (précisez en une phrase où se situe la différence).

**Réponse:** Dans les langages à portée lexicale, l'identité d'un identificateur non local est déterminée à la compilation en remontant lexicalement dans l'environnement le plus directement englobant où l'identificateur a été déclaré. En revanche, dans les langages avec portée dynamique, l'identité d'un identificateur non local est déterminée en remontant le chaînage des appels et ne peut donc se faire qu'à l'exécution.

Voici un exemple de code (avec une syntaxe C) où l'exécution de la fonction `main` provoquera l'affichage de 2 dans le cas d'un langage à portée dynamique (la valeur de `a` dans `main`) et de 3 dans le cas d'un langage à portée lexicale (la valeur de la variable globale).

```
int a = 3;  // variable globale

void fct() {
  printf(``%d\n'',a);
}

void main() {
  int a = 2;
  ftc();
}
```

**A-2** Soit le pseudo-code suivant déclarant trois variables globales $i$ (entière), $n$ (entière) et $t$ (tableau de d'entiers) et la fonction *maFonction* où $x$ est une variable de type entier passée en paramètre par **nom**:

```
integer i,n;
integer t: array(1:10);

maFonction(x): name x, integer x;
begin
  x = x + x;
  i = x + 1;
  x = x - i;
end
```

**A-2.1** Que valent $n$ et $i$ après l'appel qui suit ?

```
n = 1; i = 1;
maFonction(n);
```

**Réponse:** $n$ vaut -1 et $i$ vaut 3

**A-2.2** Qu'aurait valu $n$ après l'appel précédant si le passage de paramètre avait eu lieu par **référence** ?

**Réponse:** $n$ vaudrait -1 et $i$ vaudrait 3 (aucun changement)

**A-2.3** Que valent $i$ et $t$ après l'appel qui suit (pour $t$, affichez dans l'ordre les 10 valeurs $t(1) \ldots t(10)$)?

```
for i := 1 step 1 until 10 do t(i) = i; // init de t(i) a i (i in [1,10])
i = 1;
maFonction(t(i));
```

**Réponse:** $i$ vaut 3 et $t$: 2 2 0 4 5 6 7 8 9 10

**A-2.4** Qu'auraient valu $i$ et $t$ après l'appel précédant si le passage de paramètre avait eu lieu par **référence** ?

**Réponse:** $i$ vaudrait 3 et $t$: -1 2 3 4 5 6 7 8 9 10

**A-3** Écrire une **grammaire** qui reconnaît les chaînes sur l'alphabet $a, b$ formées d'un nombre identique de a et de b (et seulement ces chaînes). La chaîne vide appartient au langage. Exemple de chaînes appartenant au langage: $ab, ba, abba, baba, bbaa, \ldots$

**Réponse:**

$$\begin{array}{lll} \mathrm{S} \rightarrow \mathrm{aB} & \mathrm{S} \rightarrow \mathrm{bA} & \mathrm{S} \rightarrow \epsilon \\ \mathrm{B} \rightarrow \mathrm{Sb} & \mathrm{B} \rightarrow \mathrm{bS} & \\ \mathrm{A} \rightarrow \mathrm{Sa} & \mathrm{A} \rightarrow \mathrm{aS} & \end{array}$$

Une solution encore plus simple:

$$\mathrm{S} \rightarrow \mathrm{aSbS} \quad \mathrm{S} \rightarrow \mathrm{bSaS} \quad \mathrm{S} \rightarrow \epsilon$$

**A-4** À votre avis ce langage est-il régulier (type 3) ou pas ? Justifiez sans le démontrer votre réponse.

**Réponse:** Ce langage ne peut être de type 3, car une grammaire régulière ne permet pas de mémoriser des comptes (ici le nombre de $a$ ou de $b$). C'est cependant un langage de type 2 (la grammaire donnée est de type 2).

**A-5** Dans le contexte de `Scheme`, définissez la notion de *continuation*. Donnez un exemple de programme illustrant ce concept et donnez un exemple d'utilisation ainsi que le résultat de votre programme.

**Réponse:** Une continuation est une fonction $f$ passée à une autre fonction $g$ et qui doit être appliquée à toutes les valeurs retournées par $g$. L'exemple ci-dessous applique une continuation *cont* à la fonction exemple (ici l'identité).

```
(define exemple (lambda (x cont) (cont x)))
```

L'appel suivant sera évalué à 5 (application de la continuation "ajout de 3" au paramètre 2 passé à `exemple`):

```
(exemple 2 (lambda (x) (+x 3)) )
```

**Question B:** *Le coin:* `C` (20 points)
Écrivez un programme C (alternativement C++ si vous n'utilisez pas la Standard Template Library (STL)) qui calcule la valeur d'une expression arithmétique pré-fixée mettant en jeu les seuls opérateurs + et * d'arité 2. Dans les expressions que vous devez analyser, il peut y avoir un nombre quelconque (éventuellement vide) d'espace entre un opérateur et ses opérandes (c'est-à-dire: `+3*4+24 1` et `+ 3 * 4 + 24 1` désignent la même expression).
Voici des exemples d'appels de votre programme et la valeur de retour

```
calcul ``+ 3 * 4 + 2 1''  15
calcul ``+ 3 2''          5
calcul ``* + 3 * 2 7 4''  68
```

**Conseil:** Il vous appartient de gérer correctement la ligne de commande. Si vous n'en êtes pas capable[1], mais que vous traitez tout de même l'exercice, commentez intelligemment votre code pour que je puisse le comprendre !

**Réponse:** Allez voir le solutionnaire de la démonstration 3 pour une solution complète, voici un exemple de code qui répond cependant au problème posé:

```
#include <iostream>
#include <stdlib.h>
#include <string>
#include <ctype.h>

inline void Error() {
  cerr << "Erreur de syntaxe dans l'expression" << endl;
  exit(0);
}

#define INT 1
#define PLUS 2
#define MULT 3
#define EOFT 4

int next(); // le mini analyseur lexical (voir plus bas)
char *calu,*blanc;
string lexeme; // le dernier lexeme analyse (si c'est un nombre)

int Interpreteur() { // le mini interpreteur (pile inutile)
  switch (next()) {
  case PLUS:  return Interpreteur() + Interpreteur();
  case MULT:  return Interpreteur() * Interpreteur();
  case INT:   return atoi(lexeme.c_str());
  }
}
```

[1]Ce qui signifierait entre-autre que vous n'avez pas fait vous même le TP1, ni même les démonstrations, il s'agit à toutes fins utiles d'un exercice dont la solution vous a été donnée...

```
int next() { // le mini analyseur lexical
  static bool pasInit = false;          // on fait avancer le ptr sur la chaine
  if (pasInit) calu++;                  // sauf la premiere fois

  pasInit = true;
  lexeme = "";                          // vidons le lexeme (eventuel) precedant

  for (; (*calu) && isspace(*calu); calu++); // sautons les blancs (eventuels)
  if (*calu == 0) return EOFT;          // rien dans l'expression (sauf des blancs)

  if (*calu == '+') return PLUS;        // on vient de reconnaitre le +
  if (*calu == '*') return MULT;        // on vient de reconnaitre le *

  for (; (*calu) && isdigit(*calu); calu++) // tant que chiffres
    lexeme = lexeme + (*calu);

  if (lexeme.size() == 0) Error();       // on attendait au moins un chiffre
  if (lexeme.find_first_not_of("0123456789") == string::npos) return INT;

  Error();
}


int main(int argc, char **argv) {
  if (argc != 2) {               // verif que argv[1] existe
    cerr << "Syntaxe attendue: " << argv[0] << " <expression>" << endl;
    exit(0);
  }
  calu = argv[1];

  // lance le calcul
  cout << argv[1] << " -> " << Interpreteur() << endl;

  // verifie que calu pointe sur une fin de ligne ``vide''
  for (; (*calu) && isspace(*calu); calu++);
  if (*calu) Error();

  return 0;
}
```

**Question C:** *Le coin: Cours bis* (10 points)
Soit les macros `echange` et  et la méthode `proc` suivantes:

```
#define echange(x,y) { int t = y; y = x; x = t; }
#define egal(t,a,b) (t[a++] == t[--b])

void proc (int a, int b, int t, int t2){
 echange(a,b);
 printf(``A=%d B=%d\n'',a,b);
 echange(t,t2);
 printf(``T=%d, T2=%d\n'',t,t2);
}

int main() {
  int a=10, b=5, c=2, d=3;
  proc(10,5,2,3);
  printf(``A=%d B=%d C=%d D=%d\n'',a,b,c,d);
  return 0;
}
```

**C-1** Indiquez ce que sera affiché à l'exécution de ce programme.

**Réponse:**

```
A=5 B=10
T=2, T2=3
A=10 B=5 C=2 D=3
```

**C-2** Indiquez ce que sera affiché à l'exécution de:

```
int t[10],a=1,b=2;

for (int i=0; i<10; i++) t[i] = 10-i;
if egal(t,a,b)
  printf(``t[%d]=%d t[%d]=%d\n'',a,t[a],b,t[b]);
else
  printf(``FIN\n'');
```

**Réponse:**

```
t[2]=8 t[1]=9
```

**Question D:** *Le coin:* `Scheme` (20 points)

**D-1** Indiquez ce que retournerait à l'écran l'interpréteur *Scheme* aux invocations suivantes:

**D-1.1** (cdr '(1 2 3 4))

**Réponse:** (2 3 4)

**D-1.2** (cadr '(1 2 3 4 5))

**Réponse:** 2

**D-1.3** (cadr '(1 (2 3) 4 (5 6)))

**Réponse:** (2 3)

**D-1.4** (cons 1 (cons 2 (cons 3 '())))

**Réponse:** (1 2 3)

**D-1.5** (cons 1 2)

**Réponse:** (1 . 2)

**D-2** Indiquez ce que retournerait à l'écran l'interpréteur *Scheme* à l'invocation suivante:

```
(letrec
   ((loop (lambda (n k)
            (cond
              ( (zero? k) n )
                ( (< n k) (loop k n) )
                ( else (loop k (remainder n k)))))))
   (loop 8 12))
```

**Réponse:** 4

**D-3** Que fait la fonction précédante ?

**Réponse:** Cette fonction calcule le plus grand commun diviseur des deux entiers passés en paramètre[2].

**D-4** Écrivez une fonction (en Scheme) `scramble` qui prend en paramètre une liste d'entiers et qui retourne une liste constituée des mêmes éléments mais dans l'ordre suivant: les nombres impairs sont au début de la liste en ordre inverse de leur ordre d'apparition (le premier nombre impair de la liste retournée est le dernier rencontré), les nombres pairs sont à la fin de la liste dans leur ordre d'apparition (le premier nombre pair de la liste retourné est également le premier de la liste traitée).

Voici des exemples d'appel et la valeur retournée:

[2]Cet exemple a été vu en démonstration.

```
(scramble '(7 6 5 4 3 2 1))  retourne (1 3 5 7 6 4 2)
(scramble '(1 2 3 4 5 6 7))  retourne (7 5 3 1 2 4 6)
(scramble '(2 6 3 3 4 8))    retourne (3 3 2 6 4 8)
(scramble '(1 2))            retourne (1 2)
(scramble '(4 2))            retourne (4 2)
```

Vous devez écrire une seule méthode qui ne contient qu'un seul paramètre: la liste à réordonner. Dans l'éventualité où vous n'y arriveriez pas, proposez une solution avec deux méthodes.

**Réponse:**

```
(define scramble
    (lambda (liste)
      (letrec ((help
                 (lambda (l i p)
                   (cond ( (null? l) (append i p) )
                         ( (odd? (car l)) (help (cdr l) (cons (car l) i) p) )
                         ( else (help (cdr l) i (append p (list (car l)))) )))))
        (help liste '() '()))))
```

**Question E:** *Le coin:* `Prolog` (25 points)
Dans cette question, vous pourriez éventuellement avoir besoin des éléments suivants:

- La comparaison entre deux entiers ou atomes peut se faire à l'aide de l'opérateur `=`,
- Il existe deux prédicats: `integer` et `atom` qui sont vrais lorsque l'argument est respectivement un entier ou un atome (`atom(truc)` est vrai, `atom(2)` est faux, `integer(2)` est vrai).
- L'opérateur impur `rem` permet d'obtenir le reste de la division entière (`X is Y rem 3` dit que X est le reste de la division de Y par 3).
- Le signe % indique que ce qui suit ce symbole (jusqu'à la fin de la ligne) est un commentaire.

**E-1** Quelle est la réponse de `Prolog` aux requêtes qui suivent sachant les règles suivantes (dont la lecture se fait de la gauche vers la droite et du haut vers le bas (ordre habituel !)) ? Vous devez répondre avec précision. Si `Prolog` est amené à demander à l'utilisateur si ce dernier souhaite une autre solution, supposez que l'utilisateur repond toujours oui (en tapant `;`).

```
lien(a,b). lien(a,x). lien(a,c). lien(b,c). lien(b,d). lien(b,f). lien(d,f).
%lien(X,Y) :- lien(Y,X).

rel(X,X).
rel(X,Y) :- lien(X,Z), rel(Z,Y).
```

**E-1.1** `rel(a,f).`

**Réponse:** yes

**E-1.2** `rel(c,f).`

**Réponse:** no

**E-1.3** `rel(b,Z).`

**Réponse:**

```
Z = b ? ;
Z = c ? ;
Z = d ? ;
Z = f ? ;
Z = f ? ;
no
```

**E-1.4** On retire maintenant le commentaire devant la huitième règle. Que produit l'effacement de `rel(c,d)` ? Justifiez.

**Réponse:** Prolog boucle car il essaye d'effacer maintenant `rel(d,c)` qui à son tour implique d'effacer `rel(c,d)`, etc.

**E-2** Écrivez le prédicat `egal(L1,L2)` qui est vrai **ssi**[3] L1 et L2 sont des listes identiques (les listes L1 et L2 ne contiennent dans cette question que des entiers ou des atomes). On vous demande d'écrire ce prédicat (c'est à dire de comparer deux à deux les éléments des deux listes) et pas d'utiliser des opérateurs qui font le travail pour vous[4].

**Réponse:**

```
egal([],[]).
egal([X|TS1],[X|TS2]) :- egal(TS1,TS2).
```

**E-3** Écrivez le prédicat `transforme(L1,L2)` qui est vrai si la liste L2 est la transformée de la liste L1 selon les spécifications suivantes:

L1 est constituée d'éléments entiers ou atomiques et lorsque dans L1 un entier `e` (positif ou nul) précède directement un atome `a` alors dans L2 ces deux éléments sont remplacés par la duplication de `a` un nombre `e` de fois. Dans les autres cas, les éléments les éléments de L2 sont une recopie des éléments de L1.

Voici des exemples:

| | |
|---|---|
| `transforme([3,carotte],[carotte,carotte,carotte])` | vrai |
| `transforme([3,2,vacances,4],[3,vacances,vacances,4])` | vrai |
| `transforme([noel,2],[noel,noel])` | faux |
| `transforme([1,abricot,2],[abricot,2])` | vrai |
| `transforme([0,abricot,2],[2])` | vrai |

**Note:** vous devez définir tous les prédicats que vous utilisez.

**Réponse:**

```
fois(0,_,[]).
fois(E,A,[A|L]) :-
  E > 0,
  F is E-1,
  fois(F,A,L).

append([],L,L).
append([X|TX],L,[X|TY]) :-
    append(TX,L,TY).

transforme([],[]).
transforme([E|[A|S]],L) :-
   integer(E),
   atom(A), !,
   fois(E,A,D),
   transforme(S,LL),
   append(D,LL,L).
transforme([X|TX],[X|TY]) :-
   transforme(TX,TY).
```

[3] Si et seulement si.

[4] La solution: egal(X,X). (ou ses clônes) ne sera pas considérée.

**E-4** Écrivez un prédicat `check(L)` qui est vrai si la liste L (dont les éléments sont seulement des entiers ou des atomes) ne contient pas d'entier suivi directement d'un atome.

**Réponse:**

```
check(L) :- transforme(L,L).
```

**E-5** Écrivez un prédicat qui prend au moins deux paramètres `L` (désignant une liste d'entiers) et `X` (désignant un entier) qui est est vrai si `X` est le dernier entier pair de la liste `L`.

| | | | |
|---|---|---|---|
| Exemple: | avec `L = [1,2,3,4,5]` | et `X=2` | le prédicat ne s'efface pas |
| | avec `L = [1,2,3,4,5]` | et `X=5` | le prédicat ne s'efface pas |
| | avec `L = [1,2,6,1,4,5,7]` | et `X=4` | le prédicat s'efface |
| | avec `L = [5,2,3,4,5,7]` | et `X` non instancié | le prédicat retourne X=4. |

**Réponse:**

```
pair(X) :- Y is X rem 2, Y == 0.

dernierPair([],X,X).
dernierPair([X|L],_,Z) :- pair(X), !, dernierPair(L,X,Z).
dernierPair([_|L],Last,Z) :- dernierPair(L,Last,Z).
```

**Adieux déchirants:** Je vous souhaite de bonnes vacances et une bonne continuation. Les notes seront sur le site web dès que j'aurai corrigé les copies (ce qui prendra du temps). La permanence pour voir vos examens finaux sera également indiquée sur la page du cours.