Partiel de Compilation

Promo 2004 - Septembre 2002

Une rédaction simple mais convaincante et une mise en évidence des résultats seront très appréciées des correcteurs...

Le taux de succès au premier exercice (considéré comme trivial pour la fin d'une première année du cycle d'ingénieur EPITA) coefficiente la note totale : $N = n_1 \times (n_2 + n_3)$.

1 Contrôle de Connaissances

1. Qu'affiche le programme suivant :

```
#include <iostream>
struct Foo {
  virtual void foo () const { std::cout << "Foo::foo()" << std::endl; }
          void bar () const { std::cout << "Foo::bar()" << std::endl; }
};
struct Bar : public Foo {
  virtual void foo () const { std::cout << "Bar::foo()" << std::endl; }
          void bar () const { std::cout << "Bar::bar()" << std::endl; }
};
int main () {
  const Foo &o1 = Foo (); o1.foo (); o1.bar ();
  const Foo &o2 = Bar (); o2.foo (); o2.bar ();
  const Bar &o3 = Bar (); o3.foo (); o3.bar ();
}
```

2. Écrire un *class template* `Pair` qui stocke deux éléments, `a` et `b`, d'un même type, avec un constructeur prenant les deux valeurs.
3. Utiliser la classe précédente pour déclarer une variable `pi` qui contienne la paire $(3, 14159)$.
4. Qu'est-ce qu'un `const_iterator` dans STL ?
5. Écrire une fonction `sum` qui prenne une liste STL d'`int` par référence, et en retourne la somme. On prendra garde à la constance et au `namespace`.

2 Calcul des Liens Statiques

En Tiger, toutes les fonctions n'ont pas besoin d'un lien statique ("Static Link"), comme par exemple l'implémentation traditionnelle de `fact`.

1. Le lien statique sert à deux choses différentes en Tiger. Dans le cas le plus évident, il sert à retrouver le frame dans lequel est logée une variable non locale. Quelle est l'autre cas où l'on utilise le lien statique ?
2. Caractériser précisément les fonctions qui ont besoin d'un lien statique.
3. Donner un algorithme qui détermine l'ensemble de ces fonctions.
4. Dans le cadre du compilateur `tc` version 2004, où et comment l'implémenteriez-vous ?

3 “Displays”

Utiliser des liens statiques n’est pas la seule technologie qui permette de gérer les variables non locales, on peut également utiliser des “displays”.

Un display est un (unique) tableau des “frame pointers” des fonctions (statiquement, dans le source) parentes. Par exemple, pour :

```
let function A () =
    let function B () =
        let function C () = ...
        in C () end
    in B () end
in A () end
```

lorsque `C` est appelée par `B`, elle-même appelée par `A`, le display contient :

```
3  FP de C
2  FP du dernier B appelé
1  FP du dernier A appelé
0  FP de main
```

parce que la “profondeur statique” de `C` est 3, celle de `B` 2 etc.

Le gestion du display se fait dans les prologues et épilogues des fonctions ; pour une fonction de profondeur statique i :

```
Sauvegarder le slot i du display
Installer le FP de cette fonction dans le slot i du display
# Corps de la fonction
Restaurer le slot i du display
```

1. Pourquoi prendre la peine de sauvegarder/restaurer le display ?
2. Pourquoi prendre la peine d’installer le FP d’une fonction dans le display dès qu’on y entre, puisque l’accès à ses variables locales se fait sans l’aide du display ?
3. Soit le programme suivant :

   ```
   let var global := 69
       function global () : int = global
   in
     global ()
   end
   ```

   Montrer le code Tree (HIR) du **corps** de la fonction `global` quand on utilise les liens statiques.
4. Même question avec un display.
5. Soit une variable `x` déclarée à une profondeur d et utilisée à une profondeur u. Combien faut-il d’instructions pour accéder à cette variable en utilisant les liens statiques ?
6. Idem, en utilisant un display.
7. Combien d’instructions coûte la maintenance des liens statiques dans l’appel d’une fonction (on comptera le travail de l’appelant avant l’appel, le travail du prologue et de l’épilogue de l’appelé).
8. One more time, mais avec les displays.
9. Qu’en concluez-vous ?