Petites notes de coin de table sur setjmp

Ph. Narb

Voila les cousins des ”gotos” qui rappliquent. On ne peut les utiliser qu'à l'intérieur d'une même imbrication de fonctions. Plus précisément, il n'est possible de restaurer un contexte que s'il a été sauvegardé dans une fonction présente dans la pile des appels (dans Braquelaire, un exemple est présenté où l'on tente de restaurer un environnement dont le `setjump` est dans une fonction qui n'est plus dans la pile; un `seg. fault` sanctionne cela). Et donc leur usage ne peut être que de faire des sauts ”brutaux” dans cette imbrication, par exemple dans des appels récursifs. Ils sont aussi adaptées au traitement des erreurs.

Il y a donc deux fonctions qui permettent cela et qui proviennent de la biblio. `setjump.h`: une qui est similaire au `LABEL` et l'autre au `GOTO`. Pour la première, il s'agit de:

```
int setjmp(jmb_buf env)
```

qui permet de sauvegarder l'état du programme dans `env` où `jmb_buf` est un type qui est défini dans la bibliothèque `setjump.h`. Pour la seconde fonction, il s'agit de:

```
void longjmp(jmb_buf env, int val)
```

qui remet le programme dans l'état sauvegardé par le dernier appel à `setjump` par rapport à la variable `env`. La variable `val` devrait d'après Kernigan-Ritchie être le valeur retournée par l'appel ”indirect” à `setjmp`: il n'en est rien car même si l'on passe la valeur `0`, tout se passe bien... La valeur est transformée afin d'être différente de zéro.

Le fonctionnement est alors le suivant: un appel direct à `setjump` retourne toujours `0` et si par contre, l'appel est issu d'un `longjump`, il retourne une autre valeur. C'est pourquoi la syntaxe habituelle est de mettre le `setjump` dans une expression qui teste sa valeur. L'exemple dans Kernigan-Ritchie:

```
if (setjmp(env) == 0
    /*on continue*/
else
   /* apres un longjump */
```

Quel est exactement l'environnement que l'on retrouve après un ”saut” ?

1. Les registres machine sont les mêmes.

2. Les objets accessibles gardent les mêmes valeurs qu'au moment du `longjmp`.

3. Les variables automatiques de la fonction dans lequel se trouve le `setjmp` peuvent être indéfinies, si elles ont été modifiées depuis cet appel.

Par exemple, pour illustrer le point (2) ci-dessus:

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf context;
int i=111;

void
main(void)
{
  int j=222;
```

```
  if (setjmp(context) == 0)
    {
      printf("i:%d j:%d\n",i,j);
      i=333;  j=444;
      longjmp(context,1);
    }
  else  printf("new i:%d new j:%d\n",i,j);
}
```

L'éxecution donne bien:

```
i:111 j:222
new i:333 new j:444
```

Et donc, il n'y a pas sauvegarde de l'environnement au sens de toutes les variables visibles (comme en Scheme), et il s'agit donc bien du similaire à un `GOTO`. Le format général d'utilisation est donc:

```
#include <stdio.h>
#include <setjmp.h>
jmp_buf context;
extern void f();
extern void g();
main()
{ f();}

void
f()
{
  if (setjmp(context) == 0)
    {
      printf("one possibility \n");
      g();
    }
  else printf("another possibility \n");
}

void
g()
{
  longjmp(context,1);   /* si un truc ne va pas ou signal de fin */
}
```

Son exécution est donc:

```
one possibility
another possibility
```

Un autre cas de base:

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf context;
```

```
extern void f();
extern void g();

void
main(void)
{
  f();
}

void
f()
{
  int i = setjmp(context);
  printf("execution du corps de f \n");
  g();
}

void
g()
{
  longjmp(context,1);   /* si un truc ne va pas ou signal de fin */
}
```

Ici, l'exécution est une boucle infinie. On peut également utiliser la valeur de retour pour faire des branchements différenciés:

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf context;
extern void f();
extern void g();

void
main(void)
{
  f();
}

void
f()
{
  switch (setjmp(context))
    {
    case 0:    printf("one possibility \n");
      g();
    case 1:  printf("another possibility \n");
    case 2:  printf("yet another \n");
      break;
    default:  printf("ultimate \n");
    }
}
```

```
void
g()
{
  longjmp(context,2);
}
```

Ce programme donne bien à l'exécution:

```
one possibility
yet another
```

Mais attention si on remplace la valeur `2` par `0`, le compilateur la remplace par `1` !!!