TP6
Le débogueur `gdb` – Les trois zones de la mémoire

Langage C (LC4)

Semaine du 8 mars 2010

1 Le débogueur, `gdb`

La feuille jointe vous donne les principales commandes de `gdb`.

Question 1. Recopiez (ou téléchargez depuis la page des TP) le programme `bug.c` suivant :

```
#include <stdio.h>

void init_tab(int tab[], int taille, int val)
{
  int i;

  for(i = 0; i <= taille; i++)
    tab[i] = val;
}

void affiche_tab(int tab[], int taille)
{
  int i;

  for(i = 0; i <= taille; i++)
    printf("%d ", tab[i]);

  printf("\n");
}

int main()
{
  int n = 10;
  int tab[10];

  init_tab(tab, n, 1);
  affiche_tab(tab, n);

  return 0;
}
```

Compilez-le avec l'option `-g` de `gcc` puis exécutez-le. Si vous ne l'avez pas corrigé, il comporte un bogue (en fait deux, le même bogue dans les deux fonctions).

Question 2. Lancez le débogueur avec la commande `gdb bug` (si votre exécutable s'appelle `bug`).
- Placez un point d'arrêt au début de la fonction `main()` (`b main`);
- avancez ligne par ligne au moyen de la commande `next` (ou `n`) jusqu'à ce que le débogueur vous montre l'instruction d'appel de la fonction `affiche_tab()`;
- avancez dans l'exécution de la fonction `affiche_tab()` au moyen de la commande `step` (ou `s`).
- avancez dans l'exécution de la fonction (`n` ou `s`) jusqu'à la fin de la boucle;
- affichez la valeur de la variable `taille` (`print taille` ou `p taille`);
- puis celle de `i`, elle n'est pas celle qu'on attendait.

Question 3. Toujours dans `gdb`, avec un point d'arrêt posé sur le début de la fonction `main()`,
- relancez l'exécution du programme (`run`, ou `r`), confirmez lorsque `gdb` vous le demandera.
- Pour la deuxième exécution, vous afficherez de manière permanente le contenu de la variable `n` (`display n`) et de la première case du tableau (`display tab[0]`);
- avancez avec `s` pour entrer dans la fonction `init_tab()`;
- dans la fonction, affichez de manière permanente le contenu de `taille` (`display taille`) et de `i` (`display i`);
- avancez pas à pas jusqu'à la fin de la boucle, notez la dernière valeur de ţ.
- avancez pour sortir de la boucle, où vous retrouverez la valeur de `tab[0]` modifée.

Question 4. Corrigez le bogue, recompilez et relancez `gdb`. Avancez dans la fonction d'affichage jusqu'à l'instruction `printf("\n");`. Vous remarquerez que `printf` attend le caractère de saut de ligne `'\n'` pour afficher les valeurs.

2 Les trois zones de la mémoire

Question 5. Recopier la macro suivante.

```
#define affiche_nbr(a) printf("%p (%lu)\n", a, (unsigned long) a)
```

Elle sert à afficher un nombre entier en écriture hexadécimale (plus pratique pour repérer les puissances de 2 quand on a l'habitude) et en écriture décimale entre parenthèses (plus pratique pour faire des additions et des soustractions).
Pour afficher l'adresse d'une variable `n`, vous utiliserez l'instruction `affiche_nbr(&n)`, pour afficher l'adresse contenue dans une variable `p` de type pointeur, vous utiliserez `affiche_nbr(p)` (et bien sûr `affiche_nbr(&p)` pour afficher l'adresse de la variable `p`.)

2.1 La zone statique et la pile

Question 6. Déclarez et initialisez les variables suivantes :
- deux variables globales (c'est-à-dire en dehors de toute fonction) de type caractère;
- deux variables globales de type entier;
- deux variables globales de type pointeur (`int *` par exemple);
- deux variables de type caractère dans la fonction `main`;
- deux variables de type entier dans la fonction `main`;

– deux variables de type pointeur d'entier dans la fonction `main`.
Affichez les adresses de chacune de ces variables.
Que remarquez-vous ?

Question 7. Affichez les valeurs de `sizeof(int)`, `sizeof(char)` et `sizeof(int *)` et comparez avec le nombre d'octets qui séparent les variables déclarées à la question précédente.

2.2 Appels de fonction

Question 8. Écrivez une fonction `void f1(int *p)` dans laquelle vous déclarerez une variable de type `int`, initialisée à 7 et une de type `char`, initialisée à `'A'`. Dans cette fonction, vous afficherez :
– le contenu de l'adresse pointée par le paramètre `p`
– la valeur de `p`
– l'adresse de `p`
Après l'affichage, vous modifierez la valeur de l'entier pointé par `p` *puis* la valeur de `p` (en lui affectant la valeur `NULL`).

Question 9. Dans la fonction `main`, affectez à l'un des pointeurs d'entiers l'adresse d'une des variables entières déclarées à la question 6. Appelez la fonction `f1` dans le `main`en lui passant comme argument ce pointeur. Après l'appel, affichez l'adresse et la valeur de ce pointeur, ainsi que la valeur de l'entier pointé.
Que s'est-il passé ?

Question 10. Écrivez une fonction `void f2(int *p)` dans laquelle vous déclarerez deux variables de type `int` non initialisées. Dans cette fonction, vous afficherez :
– l'adresse du paramètre ;
– l'adresse de chacune des variables ;
– le contenu des variables.
Dans la fonction `main`, appelez cette fonction *juste* après avoir appelé `f1`.
Que remarquez-vous ?

Question 11. Considérons la fonction suivante :

```
int * f0 () {
  int i[1] = {18};
  return i;
}
```

À la lumière de la question précédente, que se passe-t-il sur le tableau renvoyé lorsqu'on appelle `f1` juste après `f0` ?

2.3 Variables statiques

Question 12. Ajoutez à `f1` une variable entière statique initialisée à 9 (`static int n = 9;`) et affichez son adresse en même temps que les autres ainsi que son contenu.
Comme dernière instruction de la fonction, vous incrémenterez la valeur de cette variable.
Où cette variable est-elle stockée ?

Question 13. Dans la fonction `f2`, appelez la fonction `f1`. Que se passe-t-il ?

2.4 Le tas

Question 14. Dans le `main`(vous pouvez entamer un autre fichier si vous voulez), affichez les adresses renvoyées par cinq appels successifs à la fonction `malloc`, pour réserver successivement 1, 2, 12, 13 puis 1 octet de mémoire.
Où sont alloués ces espaces mémoire ? Sont-ils contigus les uns aux autres ?