TP5

Langage C (LC4)

semaine du 5 mars

1 Le debugger, `gdb`

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

Question 1. Recopiez 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 bug (en fait deux, le même bug dans les deux fonctions).

Question 2. Lancez le debugger 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 debugger 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 i.
– avancez pour sortir de la boucle, où vous retrouverez la valeur de `tab[0]` modifée.

Question 4. Corrigez le bug, 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 Arguments d'un programme

Quand on exécute un programme en ligne de commande, on peut faire suivre le nom du programme de plusieurs arguments, par exemple `./programme truc bidule 18`. La fonction `int main(int argc, char *argv[])` peut récupérer ces arguments, de la manière suivante :
– `argc` contient le nombre d'arguments ;
– `argv` est un tableau de chaînes de caractères contenant ces arguments ;
– `argv[argc]` est le pointeur nul `NULL`.

Attention : `./programme` est considéré comme un argument, il apparaît donc dans `argv[0]`. De plus tous les arguments sont récupérés sous la forme de chaînes de caractères (y compris le `18` ci-dessus).

Question 5. Dans un fichier `affiche_args.c`, écrivez une fonction `int main(int argc, char *argv[])` qui affiche dans la sortie standard le message suivant : « Vous avez rentré ... arguments. Voici les arguments que vous avez rentrés : ... » . Vous séparerez les arguments par des tabulations, et vous mettrez un retour à la ligne à la fin. Faites-le tourner sur plusieurs exemples.

Solution.

```
#include <stdio.h>

int main(int argc, char * argv[]) {
```

```
        int i = 0;
        printf("Vous avez rentr'e %d arguments, les voici:", argc);
        for (; i < argc; i++) {
                printf("%s\t", argv[i]);
        }
        printf("\n");
        return 0;
}
```

2.1 Conversions

Vu que tous les arguments sont des chaînes de caractères, il va falloir les convertir selon ce qu'on veut obtenir. Regardez dans le manuel la description des fonctions `atoi` et `atof`.

Question 6. Dans un fichier `somme_reels.c`, écrivez une fonction `main` qui affiche la somme des nombres réels rentrés par l'utilisateur sous la forme « La somme des nombres rentrés est ... », et en arrondissant le résultat au centième. Que se passe-t-il si l'utilisateur ne rentre aucun argument ? Modifiez votre programme pour qu'il affiche 0 si l'utilisateur ne rentre aucun argument. Faites tourner votre programme sur plusieurs exemples, en essayant notamment de rentrer autre chose que des réels pour voir ce qui se passe.

Solution.

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

int main(int argc, char * argv[]) {
        int i;
        double somme = 0;
        if (argc >= 2)
                for (i = 1; i < argc; i++)
                        somme += atof(argv[i]);
        printf("La somme des nombres rentr'es est %.2f\n", somme);
        return 0;
}
```

Question 7. Dans un fichier `somme_entiers.c`, écrivez une fonction `main` qui calcule la somme des nombres entiers rentrés par l'utilisateur. Que se passe-t-il si l'utilisateur rentre autre chose que des entiers ? Modifiez votre programme pour que, si l'utilisateur rentre autre chose que des entiers, il affiche un message sur la sortie d'erreur standard `stderr` indiquant le numéro du premier argument erroné. N'oubliez pas qu'il y a des entiers négatifs.

Solution.

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

int erreur(int i) {
        fprintf(stderr, "L'argument %d n'est pas un entier !\n", i);
        return EXIT_FAILURE;
}

int main(int argc, char * argv[]) {
```

```
        int i, j;
        int somme = 0;
        char c;
        if (argc >= 2)
                for (i = 1; i < argc; i++) {
                        c = argv[i][0];
                        if (c != '-' && (c > '9' || c < '0')) {
                                return erreur(i);
                        }
                        j = 1;
                        while (c = argv[i][j]) {
                                if (c > '9' || c < '0')
                                        return erreur(i);
                                j++;
                        }
                        somme += atoi(argv[i]);
                }
        printf("La somme des nombres rentr'es est %d\n", somme);
        return EXIT_SUCCESS;
}
```