CSH : Initiation au C et au shell

Première année

TD 1 : Pointeurs, chaînes de caractères, mémoire

★ Exercice 1. Pour commencer simplement...

Notes aux enseignants.

- Faire un rappel sur les notions de pointeurs et adresses. Les opérateurs * et & ...
- Expliquer le passage de paramètres par valeur et par adresse.

▷ **Question 1.** Quelles sont les valeurs des variables après exécution du programme suivant ? (faire un dessin de la mémoire).

```
int f(int x, int *y, int z) {
  int a;
  x = x + *y;
  a = 3;
  *y = *y - x;
  z = x - *y;
  return z;
}

int main(int argc, char *argv[]){
  int a, b, d, e;
  int *c;

  a = 5; b = 9;
  c = &e;
  *c = 23;
  d = f(a, &b, *c);
  return 0;
}
```

▷ **Question 2.** Ecrire une fonction `void echange(...x, ...y)` qui échange les valeurs de deux entiers passés en argument et la fonction `main()` qui utilise cette fonction.

▷ **Question 3.** Pourquoi la séquence suivante est-elle incorrecte ? Proposez une correction.

```
int *f(int a){
  int val[TAILLE_MAX];
  ...
  return(val);
}
```

Réponse.

Retourner une variable locale dans une fonction ne provoque qu'un warning lors de la compilation. Une pratique dangereuse car une variable locale est normalement détruite (sa durée de vie est limitée à celle de la fonction) et ça peut provoquer un comportement imprévisible du programme difficile à détecter.

Une solution à proposer consiste en la déclaration de val comme *static* afin d'étendre sa portée à l'extérieur de la fonction. Il ne sera plus stocké sur la pile mais dans le segment données du programme au même titre que les variables globales.

Si le but étant de pouvoir modifier le contenu d'un tableau, alors il suffit de le déclarer en dehors de la fonction et de le faire passer comme argument de la fonction ...

▷ **Question 4.** Dans le manuel de référence du langage C on trouve la définition suivante :

```
char* gets(char *buffer)
```

La fonction `gets()` permet de lire une chaîne de caractères jusqu'à rencontrer le caractère $\backslash n$ ou fin de fichier ; elle recopie cette chaîne à l'adresse fournie en paramètre et remplace le caractère $\backslash n$ par le caractère de fin de chaîne $\backslash 0$.

En quoi le code suivant est-il incorrect ? Proposez une correction.

```
char *ptr;
gets(ptr);
```

Réponse.

Une solution simple est la suivante :

```
char ptr[100];
gets(ptr);
```

Il faut réserver de l'espace pour contenir ce qui sera lu avec gets. Aussi, comme vu en TP2 exo1, une déclaration *ptr d'une chaine de caractères ne permet pas de faire des modifications dessus (sauf à l'initialisation) ou que l'allocation dynamique avec malloc. Voici un exemple supplémentaire vu en cours (usage calloc au lieu de malloc) :

```
char * truc; char * machin;
truc = "chaine constante";
machin = calloc(10, sizeof(char));
printf("%s\n", truc);
printf("%s\n", machin);
*truc = 'x'; // segmentation fault
*machin = 'x';
machin[1]='y'; // ou *(machin+1)='y'; ...
```

★ **Exercice 2.** Quelles sont les valeurs des variables après chaque instruction de programme suivant ?

```
int a[] = {10, 20, 30, 40, 50};

main(){
 int i, *pi, *pk, *b[2], **pl;
 pi = &a[0];
 pk = &a[1];
 pl = &pk;
 (*pl)--;
 **pl = 0;
 b[0] = &a[4];
 b[1] = b[0];
 b[0]--;
 b[0]--;
 *(b[0]) = 3;
 pi++;
 *pi = 4;
 a[a[2]] = 1;
 for(i=0; i<=4; i++)
    printf(" a[%d] = %d ", i, a[i]);
 printf("\n");
}
```

Réponse.

Imprime : 0 4 3 1 50

★ **Exercice 3.** Commentez toutes les étapes de ce programme en décrivant le contenu des différentes variables (faire un schéma du tableau et des variables). Indiquez également ce qu'il imprime.

```
#include <stdio.h>

int main() {
  char mot[] = "VACANCE";
  char *ptr, **ptr2;

  mot[1] = 'O';
  ptr = mot + 2;
  *ptr = mot[0] + 3;
  ptr++;
  ptr2 = &ptr;
  **ptr2 = *(mot + 3);
  *(++*ptr2) = 'G';
  *(ptr + 1) = *(*ptr2 + 2);
  *(ptr + 2) = 'S';
  printf( "Nouveau mot %s \n", mot);
  *(*ptr2++) = mot[7];
  printf( "Nouveau mot %s \n", mot);
}
```

Imprime : VOYAGES VOYA

★ **Exercice 4.** On considère les déclarations suivantes d'un programme C : écrivez l'instruction correspondant au commentaire.

```
int main(int argc, char *argv[]){
  char msg[9] = "bonjour!" ;
  char *adrMsg, *ptrCar, c;
  short longueur;

/* initialise adrMsg à l'adresse du 1er caractère 'b' de msg (adresse
de début) */



/* instruction identique à adrMsg = &msg[0]; */



/* initialise ptrCar à l'adresse du dernier caractère (nul) de msg */



/* ramène le pointeur sur le caractère '!'   */



/* stocke la longueur de la chaîne "jour"   */



/* stoke le caractère pointé par ptrCar dans la variable c   */



/* décrémente ptrCar, obtient le caractère 'r' et le range dans la
variable c */



/* ramène le pointeur sur le caractère '!'   */



/* stocke le caractère pointé par ptrCar ('!') dans c, puis incrémente
ptrCar */



/*  stocke le caractère contenu par la variable c à l'adresse du
caractère 'j'  */



}
```

Réponse.

```
adrMsg = &msg[0] ou encore adrMsg = msg
ptrCar = &msg[8]
--ptrCar
longueur = (short)(ptrCar - (adrMsg + 3))
c = *ptrCar (équivaut à c = msg[7] ou c = *(adrMsg + 7) )
c = *--ptrCar
++ptrCar
c = *ptrCar++
*(ptrCar - 5) = c
```

Eléments supplémentaires

Des exemples qui peuvent aider à expliquer certains concepts.

Les chaînes de caractères.

```
char nom[50];
```

Puis rappeler le caractère NULL de fin de chaine, qu'il y a des fonctions de manipulation (strcpy(...), strcmp(), strlen(), etc) et inclure <string.h>.

Evoquer (??) aussi les fonctions `sscanf(...)` et `sprintf(...)` avec un exemple d'utilisation. Cf. les TP déjà faits.

Les pointeurs.

Dans l'instruction `int tab[10], *p;` expliquer ce que vaut `tab` et `&tab[0]`. Faire un dessin au besoin.

Expliquer la différence entre les deux séquences suivantes :

```
char nom[] = "bonjour";
char nom[] = {'b', 'o', 'n', ... 'r', '\0'};

char nom[] = "bonjour";
char *nom = "bonjour";
```

Que se passe-t-il pour ce qui suit :

```
char v1 = {'a', 'b', 'c', 'd'};
char v2[] = "abcd";
char *p = "abcd";

sizeof(v1) = ?    --> 4
sizeof(v2) = ?    --> 5
v2[0] = 'A';          OK!
*p = 'A';             seg fault !
```

Ce qui suit peut être fait à la fin du TD s'il reste du temps :

Tableau de pointeurs : on donne

```
static char *jour[] = {"lundi", "mardi", ... "dimanche"};
```

Faire un schéma, et demander ce que vaut :

```
jour[0]
*(jour[0])
*(jour[0] + 1)
```

Rappeler la différence entre les deux déclarations suivantes :

```
int *tabPtSurEntier[5];
int (*ptTabEntier)[10];
```