UNIVERSITÉ PARIS SUD
IUT D'ORSAY

DUT Informatique
M3101 Système S3
2015 / 2016

Travaux Pratiques n° 0 : Rappels de programmation C

Nom(s) :
Groupe :
Date :

Objectifs : renouer avec les notions, la manipulation et l'écriture de programmes C, en particulier le travail avec les pointeurs, l'utilisation de structures et de formats.

1 Hello world

Voici un exemple minimal de programme C qui affiche la chaîne de caractères *hello, world* (cet exemple célébrissime repris dans tous les langages a d'ailleurs été fait à l'origine en C en 1978 par les créateurs du langage, Brian Kernighan et Dennis Ritchie).

```
#include <stdio.h>
int main() {
  printf("hello, world\n");
  return 0;
}
```

Vous devrez utiliser un éditeur pour créer vos fichiers sources, qui auront l'extension `.c` pour les fichiers contenant le code C (et non `.C` en majuscule ou `.cpp` comme en C++) ou `.h` dans le cas des fichiers de *headers*. Pour compiler vos fichiers sources (par exemple `exo.c`) et créer un fichier exécutable (par exemple `exo`), placez-vous dans le répertoire dans lequel vous avez enregistré le source et utilisez la commande : `gcc -Wall -Werror exo.c -o exo`

Vous pourrez alors lancer l'exécution de votre programme en utilisant la commande : `./exo`

Créez un fichier `hello.c` contenant le code ci-dessus, compilez-le et exécutez-le.

2 Sorties en C

Le langage C ne dispose pas du flot de sortie `cout` bien connu en C++. À sa place, on utilise une fonction spécifique nommée `printf` (cette fonction est aussi bien connue en PHP). L'instruction

```
printf("La moyenne des %d entiers est %f.\n", i, moyenne);
```

affiche sur la sortie standard (l'écran) la chaîne de caractères comprise entre les guillemets (appelée un *format*) mais où `%d` a été remplacé par la valeur de la variable entière `i`, `%f` a été remplacé par la valeur de la variable réelle `moyenne` et `\n` n'a pas été affiché mais un retour à la ligne a été effectué. `%d` et `%f` sont appelés des *codes*. Ils définissent la manière dont une variable d'un certain type doit être affichée. Le $n^{\text{ième}}$ code est destiné au $n^{\text{ième}}$ paramètre après la chaîne de caractères. La carte de référence vous donne plus de détails sur les codes. Le caractère spécial `\n` désigne le retour à la ligne.

Reprenez `hello.c` pour tester différents formats à l'aide de la fiche de référence.

Que se passe-t-il s'il y a trop de codes pour pas assez de paramètres ? Et inversement ? (Pensez à ne pas utiliser l'option `-Werror` pour cet exercice.)

Que se passe-t-il si on applique un code d'un type pour un paramètre d'un autre type ? (Pensez à ne pas utiliser l'option `-Werror` pour cet exercice.)

3 Pointeurs

Ce qui fait aujourd'hui de C (et aussi de C++) le langage préféré de la performance (jeux vidéos, applications embarquées multimédia, systèmes d'exploitation, calcul scientifique etc.) est qu'il permet à l'utilisateur de gérer lui-même la mémoire [1]. On peut accèder directement à des zones de la mémoire par l'intermédiaire de leurs *adresses*. Pour connaître l'adresse d'une variable, on peut utiliser l'opérateur « & », par exemple si `num` est une variable de type entier, alors `&num` est l'adresse mémoire à laquelle se trouve cette variable. Pour connaître la taille en octets de la zone mémoire allouée à une variable, on peut utiliser l'opérateur **`sizeof`**, par exemple la place occupée par `num` est donnée par **`sizeof`**`(num)` ou encore **`sizeof(int)`**.

On appelle les variables qui contiennent les adresses des *pointeurs* (utiles par exemple pour stocker `&num`). On déclare un pointeur en ajoutant le caractère étoile « ∗ » avant le nom de la variable. Par exemple l'instruction suivante :

```
int *ptr;
```

déclare une variable nommée `ptr` destinée à contenir l'adresse de début d'une zone de mémoire contenant un entier. On dit que `ptr` est un *pointeur sur entier*. Dans le reste du programme, on utilisera `ptr` pour désigner l'adresse d'un entier (par exemple `ptr` aurait pour valeur l'adresse de `num` si on effectuait l'affectation `ptr = &num`), et `*ptr` pour désigner l'entier qui se trouve à cette adresse (par exemple pour changer la valeur de `num`, on pourrait exécuter `*ptr = 2`).

Soit le code suivant :

```
#include <stdio.h>

int main() {
  double valeur = 10;
  double *pv = &valeur;
  int nombre, *pn;
  valeur = *pv + 1;
  printf(" valeur = %f\n", valeur);
  printf("&valeur = %p\n", &valeur);
  return 0;
}
```

1. En C++, on préférera les références, les itérateurs et autres pointeurs intelligents.

Que représente `valeur` ? Qu'affiche le premier `printf` ?

Que représente `&valeur` ? Qu'affiche le second `printf` ?

Que représente `pv` ?

Que représente `&pv` ?

Que représente `*pv` ?

Quelle est la taille de la zone mémoire réservée pour `valeur` ? Pour `pv` ? Pour `nombre` ? Pour `pn` ? Pourquoi, alors que les tailles sont différentes pour `valeur` et `nombre`, les tailles sont identiques pour `pv` et `pn` ?

En C, absolument tous les paramètres des fonctions sont des *copies* des arguments passés lors des appels à ces fonctions. Cela signifie que l'on ne travaille *jamais* directement sur les éléments passés lors de l'appel, mais avec d'autres variables qui contiennent, par contre, exactement les mêmes valeurs que les originales. Soit le programme suivant :

```
#include <stdio.h>

void echange1(float a, float b) {
  float temp = a;
  a = b;
  b = temp;
}
```

```
int main() {
  float pi=2.71828, e=3.14159;
  printf("Avant echange : pi = %f, e = %f.\n",pi,e);
  echange1(pi,e);
  printf("Apres echange : pi = %f, e = %f.\n",pi,e);
  return 0;
}
```

Qu'affiche ce programme ? Expliquez.

Écrivez une nouvelle fonction `echange2` qui réalisera effectivement l'échange des valeurs grâce à des pointeurs. Par ailleurs, vous ferez en sorte que les valeurs ne s'affichent qu'avec deux décimales. **Vous joindrez le code complet commenté avec un exemple d'exécution à votre compte rendu.**

4 Types structurés

Le langage C permet avec les *structures* de désigner sous un seul nom un ensemble de valeurs pouvant être de types différents (les tableaux sont souvent suffisants pour les ensembles de valeurs d'un même type). Les constituants de la structure sont appelés des *champs*. Par exemple un produit dans un magasin peut être défini par son numéro (de type entier), son prix (de type réel) et son poids (de type réel). On définira alors ce type de la manière suivante :

```
struct produit {
  int numero;
  float prix;
  float poids;
};
```

Une fois le nouveau type défini, il devient possible d'en déclarer des variables. La zone mémoire réservée pour ces variables est égale à la concaténation des zones mémoire nécessaires aux différents champs ; ces dernières sont rangées consécutivement, et dans le même ordre que celui de la définition (ici `numero` puis `prix` puis `poids`). L'instruction suivante crée deux variables de type `struct produit` (la première est initialisée à zéro [2] et la seconde à trois valeurs données) et un pointeur sur produit :

```
struct produit livre={0}, pneu={23456,94.99,26}, *ptr;
```

L'accès à chaque élément de la structure se fera par son nom au sein de la structure grâce à l'opérateur « `.` ». Par exemple on accède au champ `prix` de la variable `pneu` par `pneu.prix`. Si dans l'exemple précédent on a affecté l'adresse de `pneu` à `ptr` par l'instruction `ptr = &pneu`, on pourra accéder classiquement au champ `prix` *via* `ptr` par `(*ptr).prix`, mais C offre aussi une notation plus pratique avec l'opérateur `->` qui s'utilise ainsi : `ptr->prix`.

Ajouter ce qu'il faut à votre programme pour créer les variables `livre`, `pneu` et `ptr`. Testez les accès (lecture, écriture) aux différents membres.

2. L'initialisation à zéro met tout champ numérique à `0`, tout champ pointeur à `NULL`, et tout champ tableau ou structure est initialisé à zéro récursivement.

Comment afficher la valeur `numero` de `livre` ?

Comment afficher l'adresse mémoire du champ `poids` de `pneu` ?

Comparez les adresses de `pneu` et des trois champs de `pneu`.

Quelle est la taille de la zone mémoire réservée à `livre`, à `pneu`, à `ptr` ?

5 Chaînes de caractères

Il n'existe pas en C de véritable type chaîne de caractères comme c'est le cas en C++ ou Java. À la place, on utilise généralement des tableaux de caractères, le dernier élément de la chaîne de caractères étant désigné par convention par le caractère spécial `'\0'`. Les instructions suivantes présentent deux cas de déclaration et d'initialisation identiques de chaînes de caractères :

```
char chaine1[20] = "coucou";
char chaine2[20] = {'c','o','u','c','o','u','\0'};
```

Attention, si cela est accepté pour l'initialisation, il n'est pas permis quand on utilise des tableaux déclarés statiquement de réaliser ensuite dans le programme une opération telle que

```
chaine1 = "aurevoir";
```

La bibliothèque standard du C propose de nombreuses fonctions utiles pour la manipulation des chaînes (voir la carte de référence et les pages de `man` pour plus d'informations). Le programme ci-après évoque certaines des plus utiles :

```
#include <stdio.h>
#include <string.h>
#define TAILLE_MAX 100

int main() {
  char chaine1[TAILLE_MAX] = "bonjour,", chaine2[TAILLE_MAX];
  printf("%s (taille chaine = %d)\n",chaine1,strlen(chaine1));
  strcpy(chaine2,chaine1);
  strcat(chaine2," monde");
  printf("%s (taille chaine = %d)\n",chaine2,strlen(chaine2));
  sprintf(chaine1,"%s mond%d",chaine1,3);
  printf("%s (taille chaine = %d)\n",chaine1,strlen(chaine1));
  return 0;
}
```

Écrivez, compilez et exécutez ce programme. Quel est le résultat ?

À quoi servent les fonctions `strlen`, `strcpy` et `strcat` ?

Décrivez l'utilisation de la fonction `sprintf`.

6 Entrées en C

Le langage C ne dispose pas du flot d'entrée `cin` bien connu en C++. À sa place, on utilise une fonction spécifique nommée `scanf`. L'instruction

```
scanf("%f %d", &x, &i);
```

lit des informations sur le fichier standard d'entrée (le clavier). Comme `printf`, `scanf` possède en premier argument un format exprimé sous la forme d'une chaîne de caractères (ici `"%f %d"`). Les arguments suivants sont les adresses où il faudra « ranger » les informations lues. Cette instruction lira donc un flottant et un entier qu'elle rangera respectivement dans les variables `x` et `i`. Pour les chaînes de caractères, l'emploi de `scanf` et du code « `%s` » est peu pratique car le caractère *espace* est considéré comme un délimiteur (pas de chaînes avec des espaces). On préférera utiliser la fonction `fgets` (appel `fgets(buffer,taille,stdin)`) qui permet de lire une ligne du fichier standard d'entrée. La fonction `fgets` place la ligne lue (avec le(s) caractère(s) de fin de ligne) dans le tampon préalloué `buffer`, la lecture s'arrête après `taille-1` caractères, et le caractère `'\0'` est toujours ajouté dans `buffer` après le dernier caractère lu.

Expérimentez diverses entrées avec `scanf` et `fgets`.