ASR TD1
Initiation au langage C

Ce TD présente quelques bases du langage C qui est un langage impératif compilé (pas d'interpréteur ou de machine virtuelle, gestion explicite de la mémoire, pas d'héritage, de polymorphisme, etc). Le langage C est donc très utilisé dans la programmation système, par exemple pour l'écriture du noyau Linux, dans le domaine de l'embarqué (programmation de micro-contrôleurs), et également pour écrire des routines de calculs efficaces. On peut appeler du code binaire généré à partir de code C depuis du code C++, Java (Java Native Interface), Python , etc.

1 Hello World!

1. Créer un fichier nommé `hello.c` et taper le code suivant :

```
1 #include <stdio.h>
2
3 int main() // Point d'entrée par défaut de tout programme C.
4 {
5    puts("Hello World!");
6    return 0; // Valeur retournée au système en sortie de programme,
7    // 0 si tout se passe bien.
8    // Pour afficher cette valeur en ligne de commande : echo $?
9 }
```

   La directive `#include` permet d'inclure le fichier `stdio.h` qui contient les routines pour les entrées/sorties dont la fonction `puts` qui permet d'afficher une chaîne de caractères sur la sortie standard. Les fonctions standards possèdent généralement une page `man` contenant leur documentation (essayer `man puts`). Le point d'entrée d'un programme C est la fonction `main` qui retourne au système un entier pour indiquer si son exécution s'est bien déroulée : 0 si pas d'erreur, une autre valeur pour indiquer un code erreur.

2. Compiler ce fichier à l'aide de la commande : `gcc -o hello hello.c`. L'option `-o` indique le nom du fichier de sortie. `gcc` est le compilateur C du projet GNU GCC `https://gcc.gnu.org/` (licence libre). D'autres compilateurs C sont également disponibles :

   - `clang`, issu du projet LLVM `https://clang.llvm.org/` (licence libre)
   - `icc`, compilateur Intel (licence commercial, version gratuite disponible pour les étudiants)

   Le processus de compilation transforme le fichier source C en langage machine directement compréhensible par le processeur. Différents compilateurs vont généralement produire des codes binaires différents, plus ou moins optimisés en taille et en temps d'exécution. On peut aussi demander au compilateur d'essayer d'optimiser le code plus efficacement en contrepartie d'un temps de compilation plus lent avec l'option `-O2` (niveau d'optimisation 2, cf. `man gcc`), ou optimiser pour une architecture de processeur spécifique avec l'option `-march=native` (détection du processeur de la machine). Il est également conseillé d'utiliser l'option `-Wall` lors de la compilation pour que le compilateur effectue plus de vérifications et affiche des avertissements lorsque le code pourrait présenter un problème.

3. Lancer le programme avec la commande : `./hello`. Pour rappel, il faut faire précéder le nom du programme de `./` car par défaut le dossier courant ne fait pas partie des emplacements par défaut contenant des exécutables (contrairement par exemple au dossier `/usr/bin`).

2 Types de base

Les types disponibles dans le langage C sont presque les mêmes que ceux disponibles dans le langage Java. En C, les entiers sont, comme en Java, des entiers relatifs (codés en C2), mais il est possible d'avoir des entiers naturels (binaires purs) en ajoutant le préfixe `unsigned` devant le type entier. Les types flottants correspondent également (codés suivant la norme IEEE754).

L'affichage est effectué à l'aide de la fonction `printf` qui prend en paramètres une chaîne de formatage et les variables à afficher (`man 3 printf` pour la documentation). Le nombre de paramètres de cette fonction est variable, il s'agit donc d'une fonction variadique.

```c
#include <stdio.h>

int main()
{
  // Types signés (C2) : les valeurs peuvent être négatives
  // Valeurs de -2^{n-1} à +2^{n-1} - 1.
  char c0 = 'a'; // 8 bits, de -128 à +127
  short s0 = -5; // 16 bits
  int i0 = -213; // 32 bits
  long long l0 = 234; // 64 bits

  printf( "%c\n", c0 );
  printf( "%hhi\n", c0 ); // half half int donc 1 octet !

  printf( "%hi %i %lli\n", s0, i0, l0 );

  // Types non signés : valeurs positives uniquement
  // de 0 à +2^n - 1
  unsigned char c1 = 65; // de 0 à 255
  unsigned short s1 = 65000;
  unsigned int i1 = 2345;
  unsigned long long l1 = 354323;

  printf( "%c\n", c1 );
  printf( "%hhu\n", c1 ); // half half int donc 1 octet !

  printf( "%hu %u %llu\n", s1, i1, l1 );

  // Types flottants
  float f0 = 3.5f; // 32 bits
  double d0 = 1234.5678; // 64 bits

  printf( "%f %lf\n", f0, d0 );

  return 0;
}
```

3 Tableaux statiques

Les tableaux statiques (taille fixée) sont stockés dans la pile du processus. Attention, la pile possède une petite taille, généralement de quelques MB (mégabytes), donc on ne peut pas y placer un tableau de quelques millions d'entiers ou plus, ou une image de 8Mpx (8 millions de pixels, chaque pixel contenant 3 composantes R, G, B codées chacun sur 1 octet/Byte, soit 24MB).

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

int main()
{
  // Allocation des tableaux dans la pile.

  // Tableau de 10 entiers non initialisé.
  int tab0[ 10 ];

  // Tableau de 8 entiers initialisé.
  int tab1[] = { 1, 2, 3, 4, 5, 6, 7, 8 };

  printf("%i\n", tab1[ 0 ]);
  printf("%i\n", tab1[ 1 ]);

  return 0;
}
```

4 Fonctions

Les fonctions C sont définies de manière presque similaire aux méthodes Java sans les préfixes `public`, `protected` ou `private` :

```c
#include <stdio.h>

void hello()
{
  puts("Hello!");
}

int plus(int a, int b)
{
  return a + b;
}

int main()
{
  hello();

  printf( "%d\n", plus( 7, 8 ) );

  return 0;
}
```

5 Pointeurs

C'est un point très délicat (surtout pour les étudiants!) du langage C. Réaliser un schéma peut vous aider à comprendre ce qui suit.

La mémoire peut être vue comme un tableau d'octets. L'adresse 10000 désigne donc l'emplacement, ou l'adresse, du 10001ème octet de la mémoire (Le premier octet se trouve à l'adresse 0). Le langage C permet de manipuler des adresses à l'aide d'un type appelé pointeur. Un pointeur contient ainsi une adresse mémoire qu'il sera possible de modifier pour se déplacer en mémoire. Un pointeur destiné à contenir l'adresse d'une variable de type `int` sera déclaré par le type `int*`.

Pour obtenir l'adresse d'une variable en mémoire, il faut la faire précéder du signe `&`. Si une variable est déclarée de la manière suivante : `int x = 5`, alors la valeur 5 est stockée dans la pile en binaire sur 4 octets (000001010000...en little endian) placés par exemple à partir de l'adresse 10000. Si un pointeur est déclaré de la manière suivante : `int* px = &x`, alors `px` est stocké dans la pile sur 8 octets (64 bits) et contient la valeur 10000.

Pour obtenir le contenu d'une adresse mémoire contenue dans un pointeur (valeur pointée pour faire court), il faut faire précéder ce pointeur du signe `*`. Par exemple, le code `*px = 6` place la valeur 6 (sur 4 octets) à l'emplacement stocké dans `px` donc à partir de l'adresse 10000.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a = 5;
6
7    printf( "contenu de a = %d\n", a );
8    printf( "adresse de a = %p\n", &a );
9
10   int* pa = &a; // pa contient l'adresse de a.
11
12   *pa = 4; // La valeur 4 est écrite à l'emplacement pointé par pa.
13
14   printf( "a = %d\n", a );
15   printf( "*pa = %d\n", *pa );
16
17   return 0;
18 }
```

6 Allocation dynamique

L'allocation dynamique de mémoire sert lorsque l'on ne connaît pas à priori la taille des données à manipuler, ou lorsque la taille de ces données dépasse la capacité de la pile. C'est par exemple le cas lorsqu'on manipule des fichiers (images, sons, ...) ou des tableaux de grandes tailles. L'allocation dynamique est plus lente que l'allocation statique car il faut effectuer un appel au système d'exploitation qui gère la mémoire. Le processus se passe de la manière suivante :

1. Demande au système d'allocation d'un nombre d'octets à l'aide de la fonction `malloc`.

2. Le système vérifie qu'il est possible d'allouer ce nombre d'octets (ou non) puis effectue l'allocation et retourne un pointeur contenant l'adresse du début de la zone allouée.

3. Le programme peut ensuite manipuler cette zone mémoire à partir de l'adresse de début. Attention à ne pas déborder du tableau sinon il se produit une erreur de segmentation car le système détecte un accès à une adresse non autorisée/hors limite.

4. Le programme doit ensuite libérer explicitement la zone allouée lorsqu'elle n'est plus nécessaire à l'aide de la fonction `free`.

En Java, l'allocation de mémoire dynamique se fait à l'aide du mot-clé `new`. Les données allouées de cette manière sont stockées dans le tas et l'adresse de ces données est stockée dans une variable. Par exemple, le code Java `int a[] = new int[10]` effectue l'allocation de 40 octets dans le tas et retourne l'adresse de début du tableau dans `a`. `a` est donc l'équivalent d'un pointeur C (Vous avez d'ailleurs sans doute déjà rencontré l'erreur Java `null pointer exception`...).

En C, l'allocation dans le tas est effectuée à l'aide de la fonction `malloc` (`man 3 malloc`) qui prend en paramètre un nombre d'**octets** à allouer et retourne l'adresse du début de la zone allouée. Attention, contrairement à Java, la mémoire du tas n'est pas gérée par un *garbage collector*, c'est l'utilisateur qui doit demander la libération de la mémoire à l'aide de la fonction `free` (`man 3 free`).

Le programme suivant présente 2 manières de déclarer une valeur entière suivant que l'on souhaite la placer dans la pile ou dans le tas :

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

int main()
{
  int a = 5; // La valeur de a est stockée dans la pile.

  // Le pointeur pa est stocké dans la pile et pointe vers une zone de
  // 4 octets allouée dans le tas.
  int* pa = ( int* )malloc( sizeof(int) );

  *pa = 3; // La valeur 3 est placée à l'emplacement indiqué par pa donc dans le tas.

  // Libération de la zone mémoire pointée par pa.
  free( pa );
  // ATTENTION, le pointeur pa n'est pas supprimé.

  return 0;
}
```

Le programme suivant présente comment déclarer des tableaux dans le tas :

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

int main()
{
  // malloc prend en paramètre le nombre d'octets à allouer dans le tas
  // et retourne l'adresse de début de la zone allouée.
  // L'adresse de retour est stockée dans un pointeur.
  // sizeof retourne la taille en octets du type passé en paramètre.
  int* tab = ( int* )malloc( 10000 * sizeof( int ) );

  // Accès à la zone mémoire allouée dans le tas.
```

```
13    *tab = 3; // 1ère valeur du tableau.
14    tab[ 0 ] = 6; // 1ère valeur du tableau, comme en Java.
15    *( tab + 10 ) = 1; // 11ème valeur du tableau, équ. à tab[10]
16    tab[ 100 ] = 123; // comme en Java !
17
18    // Parcours de la zone mémoire allouée dans le tas.
19    int i;
20    for( i = 0 ; i < 10000 ; ++i ) {
21      tab[ i ] = i; // ou *( tab + i ) = i;
22    }
23
24    // libération de la zone mémoire.
25    free( tab );
26
27    return 0;
28  }
```

La syntaxe `*(tab + i)` peut se lire comme : prendre la case mémoire contenue dans le pointeur `tab`, aller `i` cases plus loin (`tab + i`) et accéder au contenu de cette case (`*`).

7 Structures

Des variables peuvent être encapsulées dans des structures :

```
1  #include <stdio.h>
2
3  struct point2d
4  {
5    float x, y;
6  };
7
8  int main()
9  {
10   struct point2d p0 = { 1.0f, 1.0f };
11
12   p0.x += 1.0f;
13   p0.y += 2.0f;
14
15   return 0;
16 }
```

Dans l'exemple ci-dessus, le nom complet du type est `struct point2d`. Ce nom complet doit être indiqué en entier à chaque fois que l'on utilise une variable de ce type, par exemple comme paramètre de fonction, ce qui s'avère fastidieux et peu lisible. Il est possible de créer un alias pour raccourcir l'écriture en définissant le mot-clé `typedef` :

```
1  #include <stdio.h>
2
3  typedef struct
4  {
5    float x, y;
6  } point2d_t;
```

```
 7
 8  int main()
 9  {
10    point2d_t p0 = { 1.0f, 1.0f };
11
12    p0.x += 1.0f;
13    p0.y += 2.0f;
14
15    return 0;
16  }
```

Les alias définis avec un `typedef` sont généralement préfixés avec `_t`.

8 Exercices

1. Créer une fonction qui retourne la factorielle d'une valeur passée en paramètre (boucle ou récursion).
2. Lire une valeur entière au clavier à l'aide de la fonction `scanf` (`man scanf`) et afficher un message avec la valeur saisie.
3. Effectuer la saisie d'un tableau de points 2D dont la taille sera demandée à l'utilisateur.
4. Créer une fonction qui retourne le barycentre des points saisis et afficher le résultat.