Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

TP 11 | passage de paramètres

Présentation

Le but de ce TP est de manipuler et de mieux comprendre le fonctionnement des deux méthodes de passage de paramètres vues en cours : le passage par valeur et le passage par adresse.

Remarques :

- Pour répondre aux questions demandant une réponse textuelle (par opposition à un programme), donnez vos explications dans la fonction `main`, sous forme de commentaires.
- Dans les exemples de ce TP, les `??` représentent en réalité des valeurs numériques. Celles-ci ne sont pas indiquées car le but des exercices est de les deviner avant d'exécuter le programme.

1 Expérimentations

Exercice 1

Écrivez une fonction `void test1(int x)` qui reçoit un entier `x` et affiche simplement sa valeur et son adresse. Pour vérifier `test1`, effectuez les opérations suivantes dans la fonction principale `main` :

1. Déclarez et initialisez une variable `x` ;
2. Affichez sa valeur et son adresse avec `printf` ;
3. Appelez la fonction `test1` en lui passant `x` en paramètre ;
4. Affichez encore une fois la valeur et l'adresse de `x`.

exemple :

```
@main : l'adresse de x avant l'appel est 0x??????
@main : la valeur de x avant l'appel est ??
@test1 : la valeur de x est ??
@test1 : l'adresse de x est 0x??????
@main : la valeur de x apres l'appel est ??
@main : l'adresse de x apres l'appel est 0x??????
```

Remarque : pour afficher une adresse avec `printf`, on utilise le format `%p` (au lieu de `%d`).

Avant d'exécuter votre programme, tentez de deviner quelles valeurs vont être affichées à la place des `??` de l'exemple ci-dessus. Cette consigne est valable pour les autres exercices.

Comparez les adresses affichées pour `x` : comment expliquez-vous les différences observées ?

Exercice 2

On garde *exactement* la même fonction `test1`, mais cette fois on procède différemment dans la fonction `main` : au lieu de déclarer une variable `x`, on utilise une variable `n`.

exemple :

```
@main : l'adresse de n avant l'appel est 0x??????
@main : la valeur de n avant l'appel est ??
@test1 : la valeur de x est ??
```

```
@test1 : l'adresse de x est 0x??????
@main : la valeur de n apres l'appel est ??
@main : l'adresse de n apres l'appel est 0x??????
```

Qu'observez-vous ? Comparez avec l'affichage obtenu à l'exercice précédent, et donnez une justification.

Exercice 3

Écrivez une fonction `void test2(int x)` qui reçoit un entier `x`, affiche sa valeur et son adresse, *modifie* `x` de manière à la diviser par 2, puis affiche la nouvelle valeur de `x`. Dans la fonction principale `main`, effectuez les opérations suivantes :

1. Déclarez et initialisez une variable `n` ;
2. Affichez sa valeur et son adresse ;
3. Appelez la fonction `test2` en lui passant `n` en paramètre ;
4. Affichez encore une fois la valeur et l'adresse de `n`.

exemple :

```
@main : la valeur de n avant l'appel est ??
@main : l'adresse de n avant l'appel est 0x??????
@test2 : la valeur de x est ??
@test2 : l'adresse de x est 0x??????
@test2 : la valeur de x apres la division est ??
@main : la valeur de n apres l'appel est ??
@main : l'adresse de n apres l'appel est 0x??????
```

Exercice 4

Écrivez une fonction `int test3(int x)` qui reçoit un entier `x`, affiche sa valeur et son adresse, calcule `x` divisé par 2, affiche le résultat de ce calcul et *renvoie ce résultat par valeur*. Dans la fonction principale `main`, effectuez les opérations suivantes :

1. Déclarez et initialisez une variable `n` ;
2. Affichez sa valeur et son adresse ;
3. Appelez la fonction `test3` en lui passant `n` en paramètre, et utilisez la valeur renvoyée par `test3` pour mettre `n` à jour ;
4. Affichez encore une fois la valeur et l'adresse de `n`.

exemple :

```
@main : la valeur de n avant l'appel est ??
@main : l'adresse de n avant l'appel est 0x??????
@test3 : la valeur de x est ??
@test3 : l'adresse de x est 0x??????
@test3 : le resultat de la division est ??
@main : la valeur de n apres l'appel est ??
@main : l'adresse de n apres l'appel est 0x??????
```

Exercice 5

Écrivez une fonction `void test4(int x, int* resultat)` qui reçoit un paramètre passé par valeur `x` et un paramètre passé par adresse `resultat`. Cette fonction doit effectuer les opérations suivantes :

1. Afficher la valeur et l'adresse de `x` ;
2. Afficher l'adresse correspondant à `resultat` ainsi que la valeur située à cette adresse ;
3. Diviser `x` par 2 et mettre le résultat à l'adresse indiquée par `resultat` ;
4. Afficher de nouveau la valeur et l'adresse de `x` ;
5. Afficher de nouveau l'adresse correspondant à `resultat` et la valeur située à cette adresse.

Remarque : attention, le paramètre `resultat` est lui-même une adresse, il ne s'agit pas d'une variable classique.

Dans la fonction principale `main`, effectuez les opérations suivantes :

1. Déclarez et initialisez une variable `n` ;
2. Déclarez une variable `r` (il n'est pas obligatoire de l'initialiser) ;
3. Affichez les adresses et les valeurs de ces variables ;
4. Appelez la fonction `test4` en lui passant `n` et l'adresse de `r` en paramètres ;
5. Affichez encore une fois les valeurs et adresses de `n` et `r`.

exemple :

```
@main : la valeur de n avant l'appel est ??
@main : l'adresse de n avant l'appel est 0x??????
@main : la valeur de r avant l'appel est ??
@main : l'adresse de r avant l'appel est 0x??????
@test4 : la valeur de x est ??
@test4 : l'adresse de x est 0x??????
@test4 : l'adresse indiquee par le parametre resultat est 0x??????
@test4 : la valeur situee a cette adresse est ??
@test4 : la valeur de x apres la division est ??
@test4 : l'adresse de x apres la division est 0x?
@test4 : l'adresse indiquee par resultat apres la division est 0x??????
@test4 : la valeur situee a cette adresse apres la division est ??
@main : la valeur de n apres l'appel est ??
@main : l'adresse de n apres l'appel est 0x??????
@main : la valeur de r apres l'appel est ??
@main : l'adresse de r apres l'appel est 0x??????
```

Expliquez pourquoi il est inutile d'initialiser la variable `r` dans la fonction `main`.

Comparez l'adresse de `r` dans la fonction `main` et celle indiquée par le paramètre `resultat` dans la fonction `test4` : qu'observez-vous (justifiez) ?

Exercice 6

Écrivez une fonction `void test5(int* x)` qui reçoit un paramètre passé par adresse `x`, et effectue les opérations suivantes :

1. Afficher l'adresse correspondant à `x` ainsi que la valeur située à cette adresse ;
2. Modifier cette valeur en la divisant par 2 ;
3. Afficher de nouveau l'adresse correspondant à `x` et la valeur située à cette adresse.

Dans la fonction principale `main`, effectuez les opérations suivantes :

1. Déclarez et initialisez une variable `n` ;
2. Affichez l'adresse et la valeur de cette variable ;
3. Appelez la fonction `test5` en lui passant l'adresse de `n` en paramètre ;
4. Affichez encore une fois la valeur et l'adresse de `n`.

exemple :

```
@main : la valeur de n avant l'appel est ??
@main : l'adresse de n avant l'appel est 0x??????
@main : la valeur de r avant l'appel est ??
@main : l'adresse de r avant l'appel est 0x??????
@test4 : la valeur de x est ??
@test4 : l'adresse de x est 0x??????
@test4 : l'adresse indiquee par le parametre resultat est 0x??????
@test4 : la valeur situee a cette adresse est ??
@test4 : la valeur de x apres la division est ??
@test4 : l'adresse de x apres la division est 0x?
@test4 : l'adresse indiquee par resultat apres la division est 0x??????
@test4 : la valeur situee a cette adresse apres la division est ??
@main : la valeur de n apres l'appel est ??
@main : l'adresse de n apres l'appel est 0x??????
@main : la valeur de r apres l'appel est ??
@main : l'adresse de r apres l'appel est 0x??????
```

Est-il nécessaire d'initialiser la variable `n` dans la fonction `main` ?

Comparez l'adresse de `n` dans la fonction `main` et celle indiquée par le paramètre `x` dans la fonction `test5` : qu'observez-vous (justifiez) ?

2 Fonctions mathématiques

Exercice 7

Écrivez une fonction `vabs` qui calcule et retourne la valeur absolue d'un réel `x` de type `float`. La fonction doit recevoir `x` par valeur et renvoyer sont résultat par valeur.

N'oubliez pas de tester votre fonction à partir de la fonction `main`. Cette remarque est valide pour tous les exercices.

Exercice 8

Écrivez une fonction `distance`, qui prend en paramètres deux réels de type `float` `x` et `y`, et calcule puis retourne leur distance. Cette fonction doit utiliser la fonction `vabs` précédente. Cette fois, on veut que le résultat de la fonction soit passé par adresse, sous forme d'un 3ème paramètre `res`.

Exercice 9

On veut écrire une fonction `division_entiere` qui calcule et renvoie à la fois le quotient `q` et le reste `r` de la division de deux entiers `x` et `y`. Doit-on utiliser un passage de paramètre par valeur ou par adresse ? Pourquoi ? Écrivez et testez la fonction.

Exercice 10

On veut résoudre une équation du 2ème degré. Pour cela, on veut écrire une fonction `calcule_racines` qui calcule le discriminant du polynôme et l'utilise pour déterminer l'existence de solution(s), puis pour les calculer si c'est possible.

Écrivez une fonction qui reçoit par valeur les trois coefficients `a`, `b` et `c` du polynôme. La fonction doit renvoyer deux sortes de résultats :

- Elle doit renvoyer *par valeur* un code indiquant s'il existe une racine (code 1), deux racines (code 2) ou pas de racine du tout (code 0).
- Si une ou plusieurs racines existent, elle doit la (ou les) renvoyer *par adresse*.

Donc, la fonction a besoin de deux paramètres supplémentaires `r1` et `r2`, qui ne seront pas obligatoirement utilisés (ça dépend du discriminant). Tout l'affichage doit être effectué dans la fonction de l'exercice suivant, et non pas dans `calcule_racine`.

Remarque : La fonction calculant la racine carrée d'un réel `x` en langage C est `sqrt(x)`. Elle est contenue dans la librairie `math.h`, qui devra donc être inclue dans votre programme. Il est également nécessaire de modifier un paramètre de compilation. Dans les propriétés du projet Eclipse, allez dans *C/C++ Build* puis *Settings*, puis *Tool Settings*, puis *C Linker*, puis *Libraries*. À droite, dans Libraries (-l), ajoutez une librairie simplement appelée `m` (m comme mathématiques).

Exercice 11

Écrivez une fonction `void affiche_racines(float a, float b, float c)` qui reçoit les 3 coefficients d'un polynôme du 2ème degré, qui utilise `calcule_racines` pour calculer ses racines et qui affiche le résultat *exactement* comme indiqué ci-dessous.

exemple 1 : pour l'appel `affiche_racines(10, -4, 1)` on obtient l'affichage :

```
Traitement du polynome 10.00x^2 - 4.00x + 1.00 = 0
Il n'existe aucune racine relle pour ce polynome
```

exemple 2 : pour l'appel `affiche_racines(9, 12, 4)` on obtient l'affichage :

```
Traitement du polynome 9.00x^2 + 12.00x + 4.00 = 0
```

```
Il n'existe qu'une seule racine reelle pour ce polynome : r=-0.67
```

exemple 3 : pour l'appel `affiche_racines(3, -5, 2)` on obtient l'affichage :

```
Traitement du polynome 3.00x^2 + -5.00x + 2.00 = 0
Il existe deux racines reelles pour ce polynome : r1=0.67 et r2=1.00
```