Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

TP 03 | variables & opérateurs

Présentation

Le but de ce TP est de comprendre comment les variables sont placées en mémoire, et de manipuler les opérateurs de base du C.

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 Variables & adresses

Exercice 1

Écrivez un programme qui déclare :

- Trois variables *globales* `a`, `b` et `c` de type `short` ;
- Trois variables *locales* à la fonction `main` `d`, `e` et `f` de type `short`.

Dans la fonction `main`, affichez les adresses de ces six variables, en utilisant le format pointeur `%p` dans `printf`, pour afficher une adresse.

exemple : le programme doit afficher des valeurs de la forme suivante (où les `??` correspondent à des valeurs connues seulement à l'exécution) :

```
&a:0x?????? &b:0x?????? &c:0x??????
&d:0x?????? &e:0x?????? &f:0x??????
```

En vous aidant des adresses affichées par votre programme, faites un schéma de l'occupation de la mémoire par ces variables. Qu'observez-vous ?

Exercice 2

Ajoutez une fonction dans votre programme, appelée fonction :

```
void fonction()
{   ...
}
```

Vous devez la compléter en remplaçant les `...` par la déclaration de trois variables `g`, `h`, et `i` de type `short`, et en affichant leurs adresses.

Dans la fonction `main`, après l'affichage de `d`, `e` et `f`, effectuez un appel à la fonction précédente en insérant l'instruction suivante :

```
    fonction();
```

Toujours dans la fonction `main`, après cet appel, créez trois variables locales `g`, `h`, et `i` de type `short`, et affichez leurs adresses.

En utilisant les adresses affichées par le programme, complétez le schéma de l'exercice précédent. Qu'observez-vous ?

2 Opérateurs & transtypage

Exercice 3

Écrivez un programme qui saisit un caractère. Si le caractère est une majuscule, il doit être transformé en minuscule. Sinon, le caractère ne doit pas être modifié. Enfin, le caractère

obtenu doit être affiché. Vous ne devez *pas* utiliser de *valeur entière littérale* (i.e. aucun nombre ne doit apparaître dans le programme).

exemple : si l'utilisateur saisit le caractère `V` :

```
Entrez le caractere a traiter : V
Caractere apres le traitement : v
```

Remarque : on ne considère que les majuscules sans accent (é), cédille (ç), tréma (ö), etc.

Exercice 4

Écrivez un programme qui saisit deux entiers puis calcule et affiche le résultat de la division **entière**.

exemple :

```
Entrez le premier operande : 17
Entrez le second operande : 5
17 = 5*3 + 2
```

Exercice 5

Écrivez un programme qui saisit deux entiers puis calcule et affiche le résultat de la division *réelle*.

exemple :

```
Entrez le premier operande : 10
Entrez le second operande : 4
10/4 = 2.250000
```

Exercice 6

Écrivez un programme qui :

- Demande à l'utilisateur de saisir une valeur réelle ;
- Tronque cette valeur de manière à obtenir au maximum deux chiffres après la virgule ;
- Affiche le résultat de la troncature.

exemple :

```
Entrez le nombre reel : 1.234
Resultat de la troncature : 1.230000
```

Remarque : il ne s'agit pas de simplement limiter le nombre de chiffres **affichés**, mais bien de modifier la **valeur** de la variable. Vous devez utiliser le *transtypage explicite*.

Exercice 7

Analysez le code source suivant, et essayez de prévoir ce qui sera affiché à l'écran. Puis, recopiez-le dans votre programme, exécutez-le et vérifiez vos prédictions.

```
int main()
{  int x=5,y=15;
   float u=2.1,v=5.0;
   printf("x==x : %d\n",x==x);
   printf("x==y : %d\n",x==y);
   printf("x==u : %d\n",x==u);
   printf("x==v : %d\n",x==v);
   printf("x>4 || x<3 : %d\n",x>4 || x<3);
   printf("x>4 && x<3 : %d\n",x>4 && x<3);
   if(x)
      printf("(x) : vrai\n");
   else
      printf("(x) : faux\n");
   printf("x=7 : %d\n",x=7);
   printf("x : %d\n",x);
   printf("x=(7!=8) : %d\n",x=(7!=8));
   printf("x : %d\n",x);
   printf("(float)x : %d\n",(float)x);
}
```

Que pouvez-vous déduire en ce qui concerne la valeur d'une expression d'affectation ?