TP2 : Vie et mort des variables en mémoire

Exercice 1 : Modèle de pile pour l'adressage

Reprenez le fichier `Vecteur3D.cpp` que vous avez créé au TP précédent et ajoutez-y des « cout » pour visualiser les adresses mémoires des paramètres et des variables locales des différents sous-programmes.

Aide : pour afficher l'adresse d'une variable, vous pouvez la présenter en tant que long int :

```
cout << "Adresse de monFloat : " << (long int) &monFloat;
```

Vous pouvez aussi demander la taille en octets occupée par une variable ou un type avec l'opérateur sizeof :

```
cout << "Taille occupée par monTab : " << sizeof(monTab);
```

Comparez l'évolution théorique de la pile avec ce qui se passe en réalité :

a. Dans quel ordre sont empilés les éléments d'un tableau ? La case 0 a-t-elle l'adresse la plus haute ou la plus basse ?

b. Dans quel ordre sont empilés les paramètres d'une fonction ou d'une procédure ?

c. Dans quel ordre sont empilées les données membres d'un objet ?

d. Les variables locales d'un même sous-programme sont-elles, comme en TD, dans l'ordre dans lequel elles étaient déclarées dans le code ?

e. Lorsque l'un des paramètres d'un sous-programme est un tableau, combien d'octets ce paramètre occupe-t-il dans la frame du sous-programme ? Le tableau est-il recopié dans cette frame ?

f. Quel écart observez-vous entre l'adresse la plus haute et l'adresse la plus basse, parmi les adresses affichées ? Cet écart correspond-il à l'écart théorique (celui de votre trace « papier ») ? Demandez à votre encadrant d'où vient cette différence.

Exercice 2 : Modèle de pile pour les appels récursifs et visualisation des frames avec gdb

a. Tapez le programme ci-dessous qui calcule le nombre de combinaisons de p parmi n dans un nouveau fichier. Compilez-le et exécutez-le pour vérifier que tout fonctionne bien.

```
1  #include <iostream>
2  using namespace std;
3
4  /* Calcul d'un coefficient binomial à l'aide du triangle de Pascal */
5  int comb(int n, int p) {
6    int tmp1, tmp2;
7    cout <<"Calcul du nb de combinaisons de "<<p<<" elts parmi "<<n<<endl;
8
9    if ((p==0) || (n==p))
10     return 1;
11   tmp1 = comb(n-1, p-1);  /* premier appel récursif */
12   tmp2 = comb(n-1, p);    /* second appel récursif */
13   return tmp1 + tmp2;
14 }
15
16 int main() {
17   int c;
18   c = comb(4, 3);
19   cout << "c vaut " << c << endl;
20   return 0;
```

```
21 }
```

b. Puis recompilez-le en ajoutant à GCC l'option `-g` (cette option ajoute à l'exécutable des informations de débogage), et lancez le débugger gdb avec la commande `gdb nomDeVotreExecutable`. Placez un point d'arrêt sur la ligne 10 (correspondant au `return 1;`) en tapant `break 10`. gdb vous informe que ce breakpoint est le numéro 1, on pourrait en mettre d'autres si on le souhaitait. Lancez ensuite l'exécution du programme en tapant `run` ou juste `r`. L'exécution va s'arrêter lorsqu'on va entrer dans le if, juste avant d'exécuter le return.

c. Demandez à gdb la liste des « frames » actives en mémoire à ce moment en tapant `backtrace`. Combien de frames voyez-vous pour comb ? Que se passe-t-il donc quand une fonction s'appelle elle-même : réutilise-t-on la même frame ou en empile-t-on une nouvelle à chaque appel ?

d. gdb identifie les frames par des numéros (frame #0, frame #1, etc…). Sélectionnez la frame 0 pour l'examiner, en tapant `select-frame 0`. Demandez ensuite à gdb quels sont les paramètres de cette frame et quelles sont les valeurs courantes de ses variables locales en tapant `info args` puis `info locals`. Pourquoi les valeurs de tmp1 et tmp2 sont-elles étranges ?

e. Recommencez pour une autre frame. Les valeurs des paramètres sont-elles les mêmes pour des frames différentes d'une même fonction ?

f. Quittez gdb en tapant `quit`.

Exercice 3 : Allocation dynamique de mémoire dans le main

Ecrivez dans un nouveau fichier un programme qui demande à l'utilisateur de taper la taille qu'il souhaite pour son tableau, alloue un tableau de réels de la taille demandée, et demande à l'utilisateur les valeurs des réels à stocker dans le tableau. Pour la lecture des saisies clavier, vous pourrez utiliser cin. Le programme affichera ensuite le tableau et se terminera proprement, c'est-à-dire en libérant la mémoire allouée dynamiquement.

Exercice 4 : Trois entêtes pour une même fonction

Reprenez la fonction comb de l'exercice 2. Vous allez faire deux implémentations supplémentaires de cette fonction de combinaison qui vont différer par leurs entêtes, c'est-à-dire par la façon dont les entrées et les sorties sont gérées.

a. Dans le fichier écrit à l'exercice 2, indiquez les pré- et post-conditions et résultat en commentaires de la fonction déjà réalisée.

b. Ajouter une procédure de même nom (comb) qui prend en paramètre n et p mais aussi le résultat du calcul de la combinaison en mode donnée-résultat. Indiquez en commentaires les pré- et post-conditions de cette procédure. Ajouter au main un appel à cette procédure avec les mêmes valeurs de n et p que précédement et vérifier que vous obtenez le même résultat (l'afficher).

c. Ajouter une autre procédure de même nom (comb) qui prend en paramètre n et p et un pointeur sur un entier alloué sur le tas dans lequel vous mettrez le résultat du calcul. Indiquez en commentaires les pré- et post-conditions de cette procédure. Ajouter au main un appel à cette procédure avec les mêmes valeurs de n et p que précédement et vérifier que vous obtenez le même résultat (l'afficher).

Exercice 5 : Arithmétiques des pointeurs

Ecrivez un programme principal exécutant les instructions suivantes :

- Afficher la taille d'un Vecteur3D en mémoire et la taille d'un pointeur sur Vecteur3D.
- Allouer un tableau `tabVecteurPile` de 3 Vecteur3D sur la pile et un autre tableau `tabVecteurTas` de même taille sur le tas.
- Remplir les deux tableaux avec les mêmes valeurs, tirées aléatoirement entre -10.0 et 10.0 avec exactement un chiffre après la virgule.
- Afficher la taille des variables `tabVecteurPile` et `tabVecteurTas`.
- Afficher l'adresse du premier élément des deux tableaux.
- Afficher les adresses des champs x,y,z du deuxième élément des deux tableaux.
- Afficher le contenu de `*(tabVecteurPile+2)` et `*(tabVecteurTas+2)`. A quoi cela correspond-t-il ?
- Affecter au champ x du premier élément de `tabVecteurPile` la différence entre l'adresse du champ y du deuxième élément de `tabVecteurTas` et l'adresse du champ z du troisième élément de `tabVecteurTas`. Faire une version où les adresses sont converties en `long int` avant soustraction, et une version où elles ne le sont pas. Afficher et expliquer les valeurs obtenues.

Exercice 6 : Pointeurs et références

Soient les trois instructions :

```
int x = 1;
int & rx = x;
int * px = &x;
```

a. Afficher la valeur de x, la valeur de rx et la valeur de px.
b. Afficher l'adresse mémoire de la variable x, celle de rx et celle de px. Expliquer ces valeurs.
c. Ecrire trois procédures `procedureAvecPointeur (int * ptr)`, `procedureAvecReference (int & rf)` et `procedureAvecInt (int val)` qui affichent la valeur et l'adresse du paramètre.
d. Appeler ces trois procédures avec comme paramètre px pour la première, x puis rx pour la deuxième et la troisième. Expliquer les résultats affichés.