Exercices sur les pointeurs

420-C21-IN | Automne 2021, par Godefroy Borduas

Opérations élémentaires sur les pointeurs

1. Valeurs des pointeurs

```
#include <iostream>

using namespace std;

int main(void) {
      int A = 1;
      int B = 2;
      int C = 3;
      int *P1, *P2;
      P1=&A;
      P2=&C;
      *P1=(*P2)++;
      P1=P2;
      P2=&B;
      *P1-=*P2;
      ++*P2;
      *P1*=*P2;
      A=++*P2**P1;
      P1=&A;
      *P2=*P1/=*P2;
      return 0;
   }
```

À partir du tableau précédent, complétez le tableau qui suit. Dans chaque case, donnez la valeur de la variable (dans la colonne) après l'exécution de l'instruction (de la ligne).

Solution

	A	B	C	P1	P2
Initialisation	1	2	3	-	-
`P1 = &A`	1	2	3	&A	-
`P2 = &C`	1	2	3	&A	&C
`*P1 = (*P2)++`	3	2	4	&A	&C
`P1 = P2`	3	2	4	&C	&C

	A	B	C	P1	P2
`P2 = &B`	3	2	4	&C	&B
`*P1 -= *P2`	3	2	2	&C	&B
`++*P2`	3	3	2	&C	&B
`*P1 *= *P2`	3	3	6	&C	&B
`A = ++*P2**P1`	24	4	6	&C	&B
`P1 = &A`	24	4	6	&A	&B
`*P2=*P1/=*P2`	6	6	6	&A	&B

2. Résultat

À partir du code suivant, déterminer la valeur affichée pour :

a. `t = 5` b. `t = 6` c. `t = 3`

```
#include <iostream>

using namespace std;

void Foo(int t) {
   float i = 4 * t, j = t * (t - 1); // 1
   float *ptr = &i; // 2

   *ptr = j * (*ptr) + 5 * (*ptr--); // 3
   ptr = &j; // 4
   *ptr = i * t; // 5
   j = *ptr * 2; // 6
   ptr = &i; // 7
   *ptr = j / *ptr; // 8

   cout << *ptr << endl;
}
```

Solution (a : t = 5)

Ligne	i	j	ptr
1	20	20	-
2	20	20	&i
3	495	20	&i
4	495	20	&j
5	495	9900	&j

Ligne	i	j	ptr
6	495	19800	&j
7	495	19800	&i
8	40	19800	&i

Le code affichera `40`.

Solution (b : t = 6)

Ligne	i	j	ptr
1	24	30	-
2	24	30	&i
3	835	30	&i
4	835	30	&j
5	835	5010	&j
6	835	10020	&j
7	835	10020	&i
8	12	10020	&i

Le code affichera `12`.

Solution (c : t = 3)

Ligne	i	j	ptr
1	12	6	-
2	12	6	&i
3	127	6	&i
4	127	6	&j
5	127	381	&j
6	127	762	&j
7	127	762	&i
8	6	762	&i

Le code affichera `6`.

3. Valeur d'un pointeur après plusieurs fonctions

Donnez la valeur de `ptr` et de `*ptr` à la fin du code suivant :

```
#include <iostream>

using namespace std;

void Foo(int*, int);
void Bar(int*, int, int, int);

int main(void) {
   int i = 2;
   int *ptr = &i;

   Foo(ptr, i);
   Bar(ptr, 1, 2, 3);
   Foo(ptr, 0);
   Bar(ptr, 3, 2, 0);
}

void Foo(int *pointeur, int t) {
   *pointeur += t * t;
}

void Bar(int *adresse, int a, int b, int c) {
   if (c < 0)
      *adresse = (a * b + (*adresse) * 2) + c * *adresse;
   else if (c > 0)
      *adresse = (b * c + (*adresse) * 2) + b * *adresse;
   else
      *adresse *= (a + b);
}
```

Solution

| |*ptr| |Init.|2| |Foo i|6| |Bar 1, 2, 3|25| |Foo 0|25| |Bar 3, 2, 0|125|

À la fin du code, `ptr` affichera `&i` et `*ptr` affichera `125`.

4. Adresses et valeurs

Soit P un pointeur qui 'pointe' sur un tableau A:

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions:

a) `*P+2` b) `*(P+2)` c) `&A[4]-3` d) `A+3` e) `&A[7]-P` f) `P+(*P-10)` g) `*(P+*(P+8)-A[7])`

Solution

a) la valeur 14 b) la valeur 34 c) l'adresse de la composante A[1] d) l'adresse de la composante A[3] e) la valeur (indice) 7 f) l'adresse de la composante A[2] g) la valeur 23

Opérations avec tableau

5. Résultat d'une application

Qu'est-ce que le code suivant affichera ?

```
#include <iostream>

using namespace std;

int main(void) {
    int t[3];
    int i, j = 0;
    int *adt;

    for (i = 0; i < 3; i++) t[i] = j++ + i; /* 1 */

    for (i = 0; i < 3; i++) cout << t[i] << " "; /* 2 */
    cout << endl;

    for (i = 0; i < 3; i++) cout << *(t + i) - 2 << " "; /* 3 */
    cout << endl;

    for (adt = t; adt < t + 3; adt++) cout << *adt << " "; /* 4 */
    cout << endl;

    for (adt = t + 2; adt >= t; adt--) cout << *adt << " "; /* 5 */
}
```

Solution

La ligne `/* 1 */` affichera `0 2 4`. La ligne `/* 2 */` affichera `0 2 4`. La ligne `/* 3 */` affichera `-2 0 2`. La ligne `/* 4 */` affichera `0 2 4`. La ligne `/* 5 */` affichera `4 2 0` car le tableau est évalué dans l'ordre inverse.

6. Fonction de tableau

Soit deux tableaux `t1` et `t2` déclarés ainsi :

```
float t1[10], t2[10];
```

Supposons que seul le tableau t2 est initialisé avec des valeurs quelconques. Écrivez une fonction `Foo` qui prend en paramètre deux tableaux d'entier (`t1` et `t2`) sous le formalisme des pointeurs (n'oubliez pas l'indicateur de limite des tableaux). Cette fonction recopie toutes les valeurs positives de `t2` dans `t1`. Par la suite, toutes les cases qui n'auront pas été remplies dans `t1` devront être mises à 0. Enfin, imaginer l'appelle de la fonction comme suit :

```
Foo(t1, t2, 10);
```

Solution

```
void Foo(int *t1, int *t2, int NbElements) {
   int init = 0;
   for(int i = 0; i < NbElements; i++) {
      if (*(t2 + i) > 0)
         *(t1 + init) = *(t2 + i);
   }

   while(init < 10) {
      *(t1 + init++) = 0;
   }
}
```

7. Résultat d'un code de tableau

Quel sera le résultat du programme suivant :

```
#include <iostream>

using namespace std;

int main(void) {
   int t[4] = {10, 20, 30, 40};
   int *ad[4];
   int i;

   for(i = 0; i < 4; i++) ad[i] = t + i; /* 1 */
   for(i = 0; i < 4; i++) cout << ad[i] << " "; /* 2 */
   cout << endl;

   for(i = 0; i < 4; i++) cout << *ad[i] << " "; /* 3 */
   cout << endl;

   cout << *(ad[1] + 1) << " " << *ad[2] << endl; /* 4 */
}
```

Solution

`*ad[4]` est un tableau de quatre pointeurs de type entier. La ligne `/* 1 */` remplira le tableau des adresses avec cette de chaque case du tableau. La ligne `/* 2 */` affichera l'adresse des quatre case du tableau. La ligne `/* 3 */` affichera les valeurs contenues aux adresses de `ad` et par conséquent des quatre cases de `t`. La valeur affichée sera `10 20 30 40`. La ligne `/* 4 */` affichera deux fois la valeur de la troisième case de `t` car

`ad[1]` pointe sur la deuxième case de `t` et, par arithmétique des tableaux, on ajoute un élément. `ad[2]` pointe directement sur la troisième case de `t`. La valeur affichée sera `20 20`.

8 Fonctions somme

Écrivez une fonction `somme` qui renvoie la somme des éléments d'un tableau de nombre flottant. Afin de minimiser la copie en mémoire, utilisez le formalisme des pointeurs.

Écrivez un petit programme d'essai.

Solution

```
#include <iostream>

using namespace std;

float somme(float*, int);

int main(void) {
   // Le code dans Main dépend de vous.
   float Array[] = {42, 0.42, 0.0042, 0.000042};
   cout << somme(Array, 4);
}

float somme(float* Tableau, int NbElements) {
   float Resultat = 0;
   for(int i = 0; i < NbElements; i++) {
      Resultat += *(Tableau + i);
   }

   return Resultat;
}
```

9. Utilisez une structure

Prenez l'énumération et la structure suivantes :

```
enum TypeValeur_e {
   MAXIMUM, MINIMUM
};

struct ExtremeTableau_s {
   TypeValeur_e TypeExtreme;
   float Valeur;
   int Position;
   float *Reference;
}
```

Nous souhaitons créer une fonction `maxmin` qui ne retourne aucune valeur. Cette fonction aura deux tâches. Déterminer la plus grande valeur entre deux tableaux de nombre flottant `t1` et `t2` ainsi que la plus petite valeur. Les deux tableaux ont le même nombre d'éléments. Vous devrez utiliser la structure `ExtremeTableau_s` pour renvoyer la plus petite valeur et la plus grande valeur. Chaque structure doit contenir le type d'extrême, la valeur en question, la position dans le tableau et la référence vers le tableau en question. Partez du prototype suivant :

```
void maxmin(float *t1, float *t2, int NbElement, ExtremeTableau_s *max,
ExtremeTableau_s *min);
```

Solution

```
void maxmin(float *t1, float *t2, int NbElement, ExtremeTableau_s *max,
ExtremeTableau_s *min) {
   (*max).TypeExtreme = MAXIMUM;
   min->TypeExtreme = MINIMUM;
   // Les deux notations sont équivalentes

   (*max).Reference = t1;
   min->Reference = t1;
   (*max).Valeur = *t1;
   (*max).Position = 0;
   min->Valeur = *t1;
   min->Position = 0;
   for(int i = 1; i < NbElement; i++) {
      if (*(t1 + i) > (*max).Valeur) {
         (*max).Valeur = *(t1 + i);
         (*max).Position = i;
      }

      if (*(t1 + i) < min->Valeur) {
         min->Valeur = *(t1 + i);
         min->Position = i;
      }
   }

   for(int j = 0; j < NbElement; j++) {
      if (*(t2 + i) > (*max).Valeur) {
         (*max).Valeur = *(t2 + i);
         (*max).Reference = t2;
         (*max).Position = i;
      }

      if (*(t2 + i) < min->Valeur) {
         min->Valeur = *(t2 + i);
         min->Reference = t2;
         min->Position = i;
      }
```

```
    }
}
```

10. Recherche de motif ADN

Écrivez une fonction qui prend en paramètre un tableau de caractère. Vérifiez, en premier lieu, que le tableau correspond à une chaîne ADN valide (contient uniquement les lettres `A`, `C`, `G` et `T`). Dans un deuxième temps, vérifiez si le motif décrit dans un second tableau de caractère est présent dans la chaîne. Retourner `true` si le motif existe. Retournez `false` si le motif n'existe pas ou si la chaîne d'ADN n'est pas valide.

N'oubliez pas d'appliquer les principes du *Clean Code* et de vérifier si le motif est une chaîne ADN valide. Votre fonction doit utiliser uniquement des pointeurs.

Solution

Premièrement, créons une fonction qui vérifie si la chaîne demandé un code ADN valide. Ceci reste le *Clean Code* (une fonction, une responsabilité).

```
bool SequenceValide(char* sequence, int NbElement) {
   for(int i = 0; i < NbElement; i++) {
      if (*(sequence + i) != 'A' && *(sequence + i) != 'C' &&
         *(sequence + i) != 'G' && *(sequence + i) != 'T')
         return false;
   }
   return true;
}

bool Match(char *sequence, char *motif, int NbElements) {
   for(int i = 0; i < NbElements; i++) {
      if (*(sequence + i) != *(motif + i)) return false;
   }
   return true;
}

bool PossedeMotif(char *ADN, int NbMolecule, char *Motif, int NbElement) {
   if (SequenceValide(ADN, NbMolecule) && SequenceValide(Motif, NbElement)) {
      for(int i = 0; i <= NbMolecule - NbElement; i++) {
         if (Match(ADN + i, Motif, NbElement)) return true;
      }
   }

   return false;
}
```

Tableaux multidimensionnels

11. Somme 2D

Écrivez une fonction qui fournit en retour la somme des valeurs d'un tableau de nombre flottant à deux indices dont les dimensions sont fournies en argument. Utilisez le formalisme des pointeurs.

```
float somme(float *tableau, int m, int n) {
   float Resultat = 0;
   for(int i = 0; i < n * p; i++) s += *(tableau + i);
   return Resultat;
}
```