TD3: Pointeurs, pile et tas

MP2I Lycée Pierre de Fermat

Exercice 1.

Généralités

Q1. Si on a une variable `int* p`, alors de quelle type sont les valeurs `*p` et `&p` ?

Q2. Si l'on a deux variables `int x` et `int* p`, a t-on `x == *p` si et seulement si `&x == p` ? Déterminer si une des deux implications est vraie.

Exercice 2.

Appels récursifs sur la pile

On rappelle que les stack frames de la pile correspondent à des **appels** de fonction. Ainsi, lorsqu'une fonction est récursive, chaque appel aura sa stack frame correspondante.

Déterminez ce que fait le programme suivant. On pourra simuler une exécution et dessiner l'état de la pile au fur et à mesure.

```c
void f(int* x, int u, int i){
  if(i > 0)
    int t = *x;
    *x += u;
    f(x, t, i-1);
  }
}

int main(){
  int a = 1;
  int i;
  scanf("%d", &i);
  f(&a, 0, i);
  printf("%d\n", a);
}
```

Exercice 3.

Simulation pile

Pour chacun des programmes suivants, simuler l'exécution, et représenter l'état de la mémoire aux instants A, B, C, etc... indiqués en commentaire. Indiquer s'il y a des erreurs de segmentation. On représentera la pile d'appel en séparant clairement les cadres de pile des différents appels.

a)

```
1  int f(int a, int* q){
2    // B
3    *q = *q + a;
4    // C
5    q = &a;
6    // D
7  }
8
9  int main(){
10   // A
11   int x = 6;
12   int y = f(2, &x);
13   // E
14 }
```

b)

```
1  void g(int** l, int* p){
2    // C
3    **l = *p + 1;
4    *p = 0;
5    // D
6  }
7
8  void f(int a, int* q){
9    // B
10   g(&q, &a);
11   // E
12 }
13
14 int main(){
15   // A
16   int x = 6;
17   f(2, &x);
18   // F
19 }
```

c)

```
1  int* zeros(int n){
2    // B
3    assert(n < 100);
4    int t[100];
5    for (int i = 0; i < n; ++i){
6      t[i] = 0;
7    }
8    // C
9    return t;
10 }
11
12 int main(){
13   // A
14   int* t = zeros(6);
15   // D
16   printf("%d\n", t[3]);
17 }
```

Exercice 4.

Questions sur le tas

Q1. Dans le programme suivant, la variable `p` est-elle stockée dans la pile ou dans le tas ?

```
1 void main(){
2   float* p = malloc(5 * sizeof(float));
3 }
```

Q2. Dans le code suivant, la mémoire est-elle correctement libérée ? Si ce n'est pas le cas, proposer une correction.

```
1 void main()
2   int n = 100;
3   int* p = malloc(n * sizeof(int));
4   for (int i = 0; i < n; i++){
5     free(p[i]);
6   }
7 }
```

Exercice 5.

Simulation tas

Simuler l'exécution des programmes suivants, et pour chacun dessiner l'état de la pile et du tas aux instants A, B, C, D, etc... marqués en commentaire. On représentera le contenu des case mémoires non-initialisées avec des '???', et on représentera les pointeurs par des flèches. Lorsqu'une case mémoire du tas est libérée, on pourra la représenter barrée. Indiquer les fuites mémoires et les erreurs lorsqu'il y en a, et les corriger lorsque c'est possible.

a)

```c
int main(){
  // A
  int x = 5;
  int* p = malloc(sizeof(int));
  *p = x;
  // B
  x = 9;
  p = &x;
  // C
  *p = 2;
  // D
  free(p);
  // E
  return 0;
}
```

b)

```c
int**** f(){
  // B
  int**** a = malloc(sizeof(int***));
  int*** b = malloc(sizeof(int**));
  int** c = malloc(sizeof(int*));
  int* d = malloc(sizeof(int));
  int e = 52;
  // C
  *a = b;
  *b = c;
  *c = d;
  *d = e;
  // D
  return a;
}

int main(){
  // A
  int**** x = f();
  // E
  free(x);
  // F
  free(*x);
}
```

c)

```
1  /* renvoie une matrice identité de taille 3 x 3*/
2  int** grid(){
3    int** g = malloc(3*sizeof(int*));
4    // B
5    for (int i = 0; i < 3; ++i){
6      g[i] = malloc(3*sizeof(int));
7      for (int j = 0; j < 3; ++j){
8        g[i][j] = (i == j);
9      }
10   }
11   // C
12   return g;
13 }
14
15 int main(){
16   // A
17   int** g = grid();
18   int x = 7;
19   g[2][1] = x;
20   // D
21   g[1] = &x;
22   // E
23   free(g);
24 }
```

Exercice 6.

Points

On souhaite écrire un programme qui représente les points du plans comme des tableaux de taille 2 : un point (x, y) sera représenté par un tableau `float* p` avec $p[0] = x$ et $p[1] = y$.

Q1. On propose la fonction `bool egal(float* p1, float* p2)` suivante pour tester l'égalité entre deux points :

```
1 bool egal(float* p1, float* p2){
2   return (p1 == p2);
3 }
```

Expliquer pourquoi cette fonction n'est pas correcte, et dessiner l'état de la mémoire dans un exemple où elle ne fonctionne pas. Proposer une version corrigée.

Q2. Exécuter le programme suivant, identifier les fuites mémoire, et les corriger.

```
1  /* renvoie le point (x, y) */
2  float* point(float x, float y){
3    float* p = malloc(2*sizeof(float));
4    p[0] = x; p[1] = y;
5    return p
6  }
7
8  /* Renvoie le milieu de a et b, deux points */
9  float* milieu(float* a, float* b){
10   float xm = (a[0] + b[0])/2;
11   float ym = (a[1] + b[1])/2;
12   return point(xm, ym);
13 }
14
15 float main(){
16   float* a = point(3, 5);
17   float* b = point(1, 9);
18   assert(egal(milieu(a, b), point(2, 7)));
19   return 0;
20 }
```