Université de Nice-Sophia Antipolis
Polytech'Sophia

Université de Xidian
2025–2026

Examen de Langage C

Durée : 1h30
Aucun document autorisé – Mobiles interdits

Notez que les affirmations (antécédents, conséquents, rôles, et invariants) dans vos codes C entreront pour partie dans la note finale.

▶ 1. Combien de mode(s) transmission des paramètres existe(nt) en C ? Cochez une seule réponse !

- ☐ un seul, la transmission par référence.
- ☐ un seul, la transmission par valeur.
- ☐ deux, la transmission par valeur et référence.
- ☐ zéro.

Question sur 1 pt Réponse : **un seul** mode de transmission **par valeur**.

.....

▶ 2. Dans l'appel de procédure `lire(x)`, `x` est

- ☐ un paramètre *donné* ;
- ☐ un paramètre *résultat* ;
- ☐ un paramètre *donné* et *résultat*.
- ☐ zéro.

Question sur 1 pt Réponse : `x` est paramètre *résultat*.

.....

▶ 3. Écrivez en C la fonction `nbNégatifs` qui lit sur l'entrée standard `n` réels (**double**) et qui renvoie le nombre de réels négatifs lus. Cette fonction possède un seul paramètre, le nombre `n` (> 0) de réels à lire.

Question sur 3 pts

```c
/*
 * antécédent : n>0
 * rôle : lit n réels double sur l'E/S et renvoie le nombre de réels négatifs lus
 */
int nbNégatifs(const int n) {
  assert(n>0);
  int nbNég=0;
  for (int i=0; i<n; i++) {
    double x;
    scanf("%lf", &x);
    if (x<0) nbNég++;
    // invariant : nbNég est le nb de réels négatifs parmi les i premiers réels lus
  }
  return nbNég;
}
```


.....

On représente les coordonnées (x, y) d'un point du plan cartésien par un tableau de deux réels défini par la déclaration suivante :

```c
typedef double point[2];
```

Un point $p(x, y)$ sera défini par la déclaration `point p` ; tel que `p[0]`$= x$ et `p[1]`$= y$.

▶ 4. Écrivez la procédure `initPoint` qui initialise un point `p` à partir de deux coordonnées `x` et `y`. Les coordonnées et le point sont les trois seuls paramètres de la procédure.

Question sur 2 pts

```c
/*
 * antécédent : (x,y) coordonnées d'un point p du plan
 * conséquent : p=(x,y)
 */
void initPoint(const double x, const double y, point p) {
  p[0] = x; p[1] = y;
}
```

.....

▶ 5. Écrivez la déclaration de la constante `origine` qui représente le `point` de coordonnées $(0, 0)$.

Question sur 2 pts

```c
const point origine = { 0.0, 0.0 };
```

.....

▶ 6. Écrivez la procédure `écrireLnPoint` qui écrit sur la sortie standard un point `p` passé en paramètre suivi d'un *newline*.

Question sur 1 pt

```c
/*
 * rôle : écrit sur la S/S le point p
 */
void écrirePoint(const point p) {
  printf("(%lf,%lf)", p[1], p[2]);
}

/*
```

```
 * rôle : écrit sur la S/S le point p suivi d'un newline
 */
void écrireLnPoint(const point p) {
  écrirePoint(p);
  putchar('\n');
}
```

.....

► 7. Écrivez la fonction `distance` qui renvoie la distance entre deux points passés en paramètres.

Question sur 2 pts

```
/*
 * rôle : renvoie x^2
 */
double square(const double x) { return x*x; }

/*
 * rôle : renvoie la distance entre les points p1 et p2
 */
double distance(const point p1, const point p2) {
  return sqrt(square(p2[0]-p1[0])+square(p2[1]-p1[1]));
}
```

.....

► 8. On veut gérer un tableau de n points. Écrivez la procédure `initTabPoints` qui initialise un tableau `tp` de `n` (> 0) `points` initialisés de façon aléatoire. Le nombre de points `n` et le tableau `tp` sont les deux seuls paramètres de la procédure. Vous utiliserez la fonction `drand48()` qui renvoie un réel **double** sur l'intervalle $[0;1[$.

Question sur 2 pts

```
/* antécédent : n>0
 * rôle : initialise de façon aléatoire le tableau tp de n points
 */
void initTabPoints(const int n, point tp[]) {
  assert(n>0);
  for (int i=0; i<n; i++)
    initPoint(drand48(), drand48(), tp[i]);
  //
}
```

.....

► 9. Écrivez la fonction `distanceMax` qui renvoie la distance du point d'un tableau `tp` de `n` points, le plus éloigné par rapport à l'origine. Le nombre de points `n` et le tableau `tp` sont les deux seuls paramètres de la procédure.

Question sur 4 pts


```
/*
 * antécédent : n>0
 * rôle : renvoie la distance du point de tp le plus éloigné de l'origine
 */
double distanceMax(const int n, const point tp[]) {
  assert(n>0);
  double dmax = distance(tp[0], origine);
  for (int i=1; i<n; i++) {
    double dp = distance(tp[i], origine);
    if (dp>dmax) dmax = dp;
    // invariant : dmax est la distance d'un point le plus éloigné
    // des points de tp[0] à tp[i]
  }
  return dmax;
}
```

.....

► 10. Écrivez la fonction `main` qui :
- déclare et initialise une variable `n` à partir d'un entier (**int**) lu sur l'entrée standard ;
- déclare un tableau `tp` de `n` points ;
- initialise de façon aléatoire le tableau `tp`. Utilisez la procédure `initTabPoints` ;
- écrit sur la sortie standard la distance du point du tableau `tp` le plus éloigné par rapport à l'origine. Utilisez la fonction `distanceMax`.

Question sur 2 pts

```
int main(void) {
  // lire sur E/S le nb de points
  int n;
  scanf("%d", &n);
  assert(n>0);
  // déclarer et initialiser de façon aléatoire le tableau tp de n points
  point tp[n];
  initTabPoints(n, tp);
  // écrire sur la S/S la distance du point le plus éloigné de l'origine
  // contenu dans le tableau de points tp
  printf("%lf\n", distanceMax(n, tp));

  return EXIT_SUCCESS;
}
```

.....