Autotest de maîtrise du langage C

Chaque question propose plusieurs réponses possibles dont une, plusieurs ou éventuellement zéro sont correctes. Il faut cocher toutes les réponses correctes.

▶ **Q. 1** Le langage C est un langage
☐ orienté objet ☐ fonctionnel ☐ impératif ☐ typé statiquement ☐ typé dynamiquement

▶ **Q. 2** Le langage C fut développé par
☐ D. Ritchie et K. Thompson ☐ B. Kernighan et D. Ritchie
☐ B. Stroustrup ☐ B. Gates ☐ S. Jobs

▶ **Q. 3** L'instruction `n = n + 1;` est équivalente à
☐ `n++;` ☐ `++n;` ☐ `*n++;` ☐ `*++n;` ☐ `n += 1;` ☐ `n + 1 = n;`

▶ **Q. 4** La déclaration d'un pointeur `p` sur un entier se fait par
☐ `int* p;` ☐ `int p*;` ☐ `int[] p;` ☐ `int p[];` ☐ `int& p;` ☐ `int p&;`

▶ **Q. 5** Récupérer l'adresse d'une variable `n` se fait par
☐ `*n;` ☐ `n*;` ☐ `[]n;` ☐ `n[];` ☐ `&n;` ☐ `n&;`

▶ **Q. 6** La déclaration d'une fonction `f` prenant un tableau `t` d'entiers en paramètre se fait par
☐ `f(int* t)` ☐ `f(int t*)` ☐ `f(int[] t);` ☐ `f(int t[]);` ☐ c'est impossible.

▶ **Q. 7** La déclaration d'une fonction `f` retournant un tableau d'entiers se fait par
☐ `int* f()` ☐ `int[] f()` ☐ `int[f()]` ☐ `int(f[]);` ☐ c'est impossible.

▶ **Q. 8** Le type des chaînes de caractères en C est
☐ `char` ☐ `char*` ☐ `char []` ☐ `string` ☐ n'existe pas

▶ **Q. 9** Quelle syntaxe permet d'accéder à l'objet pointé par un pointeur `p` et de le modifier ?
☐ `*p = a` ☐ `p = a` ☐ `&p = a` ☐ c'est impossible

▶ **Q. 10** Quelle syntaxe permet de modifier un pointeur `p` ?
☐ `*p = a` ☐ `p = a` ☐ `&p = a` ☐ c'est impossible

▶ **Q. 11** La syntaxe `p->f()` est équivalente à
☐ `p[f()]` ☐ `*(p.f())` ☐ `*p.f()` ☐ `(*p).f()` ☐ n'existe pas

▶ **Q. 12** La syntaxe `p[n]` est équivalente à
☐ `p+n` ☐ `*(p+n)` ☐ `*p+n` ☐ `p*n` ☐ n'existe pas

▶ **Q. 13** Après la déclaration `int n = 0`, la valeur de l'expression `n = n` est
☐ `true` ☐ `false` ☐ `0` ☐ `1` ☐ `2` ☐ aucune ☐ ne compile pas

▶ **Q. 14** Après la déclaration `int n = 0`, la valeur de l'expression `n += n` est
☐ `true` ☐ `false` ☐ `0` ☐ `1` ☐ `2` ☐ aucune ☐ ne compile pas

▶ **Q. 15** Après la déclaration `int n = 0`, la valeur de l'expression `n <= n` est
☐ `true` ☐ `false` ☐ `0` ☐ `1` ☐ `2` ☐ aucune ☐ ne compile pas

▶ **Q. 16** Après la déclaration `int n = 0`, la valeur de l'expression `n == n` est
☐ `true` ☐ `false` ☐ `0` ☐ `1` ☐ `2` ☐ aucune ☐ ne compile pas

► **Q. 17** Après la déclaration `int n = 0`, la valeur de l'expression `n != n` est
☐ `true` ☐ `false` ☐ `0` ☐ `1` ☐ `2` ☐ aucune ☐ ne compile pas

► **Q. 18** Après l'exécution du fragment de code `int n = 0; n = !n;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 19** Après l'exécution du fragment de code `int n = 0; n = n++;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 20** Après l'exécution du fragment de code `int n = 0; n = ++n;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 21** Après l'exécution du fragment de code `int n = 1; int* p = &n; p++;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 22** Après l'exécution du fragment de code `int m = 1; int n = 1; int* p = &m; int* q = &n; n = p == q;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 23** Après l'exécution du fragment de code `int m = 1; int n = 1; int* p = &m; int* q = &n; n = *p == *q;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 24** Après l'exécution du fragment de code `int m = 1; int n = 1; int* p = &m; int* q = &n; n = &p == &q;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 25** Après l'exécution du fragment de code `int n = 1; int* p = &n; *p++;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 26** Après l'exécution du fragment de code `int n = 1; int* p = &n; ++*p;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 27** Après l'exécution du fragment de code `int n = 1; int* p = &(n+n); n = *p;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 28** Après l'exécution du fragment de code `int n = 1; int* p = &n; n = *p+1;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 29** Après l'exécution du fragment de code `int n = 1; int* p = &n; n = *(p+1);`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 30** Après l'exécution du fragment de code `int n = 1; n++++;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 31** Après l'exécution du fragment de code `int n = 1; int* p = &n; int** q = &p; ++**q;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 32** Après l'exécution du fragment de code `int n = 0; int m = 1; n = m = n;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 33** Après l'exécution du fragment de code `int n = 0; n = n == n;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 34** Après l'exécution du fragment de code `int n = 0; n = n++ + n++;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 35** Lors de l'exécution du fragment de code `int n = 1; f(n++, n++);`, la fonction `f` est appelée avec les valeurs
☐ `1,1` ☐ `1,2` ☐ `2,1` ☐ `2,2` ☐ imprévisible ☐ ne compile pas

► **Q. 36** La fonction `f` est définie par `int f(int n){ return n+1; }`. Après l'exécution du fragment de code `int n = f(1);`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 37** La fonction `f` est définie par `int f(int* p){ return *p+1; }`. Après l'exécution du fragment de code `int n = 1; n = f(n);`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 38** La fonction `f` est définie par `int f(int* p){ return *p+1; }`. Après l'exécution du fragment de code `int n = 1; n = f(&n);`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 39** La fonction `f` est définie par `int f(int* p){ return *p+1; }`. Après l'exécution du fragment de code `int n = f(&1);`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 40** La fonction `f` est définie par `int* f(){ int n = 1; return &n; }`. Après l'exécution du fragment de code `int n = *f();`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 41** La fonction `f` est définie par `int* f(int* p){ int n = 1; return p; }`. Après l'exécution du fragment de code `int n = 2; n = *f(&n);`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

► **Q. 42** La fonction `f` est définie par `void f(int* p, int* q){ int t = *p; *p = *q; *q = t; }`. Quels sont les appels à la fonction `f` qui échangent les valeurs des variables `m` et `n`.
☐ `f(m, n)` ☐ `f(*m, *n)` ☐ `f(&m, &n)` ☐ `f(!m, !n)` ☐ `f(m+n, m-n)`

La fonction `len` est définie de la façon suivante.

```
int len(char* s) {
  char* t = s;
  while(*s++ != '\0');
  return s-t;
}
```

► **Q. 43** Dans le fragment de code `len("abc");`, la fonction `len` retourne la valeur
☐ `-1` ☐ `0` ☐ `3` ☐ `4` ☐ imprévisible ☐ ne compile pas

▶ **Q. 44** Dans le fragment de code `char* s = "abc"; len(s);`, la fonction `len` retourne la valeur
☐ `-1` ☐ `0` ☐ `3` ☐ `4` ☐ imprévisible ☐ ne compile pas

▶ **Q. 45** Dans le fragment de code `char s[] = {'a', 'b', 'c'}; len(s);`, la fonction `len` retourne la valeur
☐ `-1` ☐ `0` ☐ `3` ☐ `4` ☐ imprévisible ☐ ne compile pas

▶ **Q. 46** Après la déclaration `int t[] = {1, 2, 3};`, le tableau `t` contient les valeurs
☐ `0, 0, 0` ☐ `1, 2, 3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 47** Après la déclaration `int[] t = {1, 2, 3};`, le tableau `t` contient les valeurs
☐ `0, 0, 0` ☐ `1, 2, 3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 48** Après la déclaration `int* t = {1, 2, 3};`, le tableau `t` contient les valeurs
☐ `0, 0, 0` ☐ `1, 2, 3` ☐ `3, 2, 1` ☐ imprévisible ☐ ne compile pas

▶ **Q. 49** Après l'exécution du fragment `int t[] = {1, 2, 3}; t = t;`, le tableau `t` contient les valeurs
☐ `0, 0, 0` ☐ `1, 2, 3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 50** Après l'exécution du fragment `int t[] = {1, 2, 3}; int n = &t == t;`, la variable `n` a la valeur
☐ `true` ☐ `false` ☐ `0` ☐ `1` ☐ imprévisible ☐ ne compile pas

▶ **Q. 51** Après l'exécution du fragment `int s[] = {1, 2, 3}; int t[] = s;`, le tableau `t` contient les valeurs
☐ `0, 0, 0` ☐ `1, 2, 3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 52** Après l'exécution du fragment `int s[] = {1, 2, 3}; int t[] = {1, 2, 3}; int n = s == t;`, la variable `n` a la valeur
☐ `0` ☐ `1` ☐ `2` ☐ `6` ☐ imprévisible ☐ ne compile pas

▶ **Q. 53** Après l'exécution du fragment `int t[] = {1, 2, 3}; *t = 3;`, le tableau `t` contient les valeurs
☐ `3, 3, 3` ☐ `3, 2, 3` ☐ `1, 3, 3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 54** Après l'exécution du fragment `int t[] = {1, 2, 3}; int* p = t; p[1] = 3`, le tableau `t` contient les valeurs
☐ `3, 3, 3` ☐ `3, 2, 3` ☐ `1, 3, 3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 55** Après l'exécution du fragment de code `int t[] = {1, 2, 3};`
`for(int i = 1; i < 3; i++) t[i] += t[i-1];` le tableau `t` contient les valeurs
☐ `1, 2, 3` ☐ `1, 3, 6` ☐ `1, 1, 1` ☐ imprévisible ☐ ne compile pas

Le tableau `t` et la fonction `sum` sont définis de la façon suivante.

```
int t[] = { 1, 2, 3, 4, 5 };
int sum(int t[], int n) {
  int s = 0;
  for (int i = 0; i < n; ++i) s += t[i];
  return s;
}
```

▶ **Q. 56** Quels sont les appels à `sum` qui calculent la somme des quatre premières valeurs du tableau `t` et retournent la valeur `10` ?

☐ `sum(t, 4)` ☐ `sum(t[1], 4)` ☐ `sum(t[0], 4)`
☐ `sum(&t, 4)` ☐ `sum(&t[0], 4)` ☐ `sum(*t, 4)`

▶ **Q. 57** Quels sont les appels à `sum` qui calculent la somme des quatre dernières valeurs du tableau `t` et retournent la valeur `14` ?

☐ `sum(t+1, 4)` ☐ `sum(&t+1, 4)` ☐ `sum(*(t+1), 4)`
☐ `sum(t[1], 4)` ☐ `sum(&t[1], 4)` ☐ `sum(*t+1, 4)`

Le type `P` est défini de la façon suivante.

```
typedef struct {
  int x;
  int y;
} P;
```

▶ **Q. 58** La déclaration d'une fonction `f` prenant en paramètre une valeur de type `P` se fait par

☐ `f(P p)` ☐ `f(P* p)` ☐ `f(P& p)` ☐ `f(P p[])` ☐ c'est impossible.

▶ **Q. 59** La déclaration d'une fonction `f` retournant une valeur de type `P` se fait par

☐ `P f()` ☐ `P* f()` ☐ `P& f()` ☐ `P[] f()` ☐ c'est impossible.

▶ **Q. 60** Après l'exécution du fragment de code `P p = {1, 2}; P q = p; int n = q.y`, la variable `n` a la valeur

☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 61** Après l'exécution du fragment de code `P p = {1, 2}; int* q = &p.x; int n = *q;`, la variable `n` a la valeur

☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 62** Après l'exécution du fragment de code `P p = {1, 2}; P* q = &p; int n = q->x;`, la variable `n` a la valeur

☐ `0` ☐ `1` ☐ `2` ☐ `3` ☐ imprévisible ☐ ne compile pas

▶ **Q. 63** La déclaration d'une fonction `F` qui prend en paramètre une fonction `f` (prenant en paramètre un entier et retournant un entier) et qui retourne un entier se fait par

☐ `int F(int f(int))` ☐ `int F(int *f(int))` ☐ `int F(int (*f)(int))`
☐ `int F(int f, int n)` ☐ c'est impossible.

▶ **Q. 64** Après les deux déclarations `typedef int (*fun)(int); fun F(fun f) {return f;}`, l'expression `F(f)(2)` est valide si `f` est défini par

☐ `int f(int n) { return n; }`
☐ `int f(int* p) { return *p; }`
☐ `int* f(int* p) { return p; }`
☐ `int* f(int n) { return &n; }`
☐ `int f() { return 0; }`
☐ ne compile pas

▶ **Q. 65** Le nombre de questions auxquelles j'ai bien répondu, y compris celle-ci, se situe entre

☐ 0 et 15 ☐ 16 et 31 ☐ 32 et 48 ☐ 49 et 64