QCM - Pointeurs

Question 1. Que renvoie l'opérateur `&` appliqué à une variable ?

A) La valeur de la variable
B) L'adresse mémoire de la variable
C) La taille en octets de la variable
D) Un pointeur nul

Question 2. Quel est le type de `p` dans la déclaration suivante ?

```
int x = 10;
int* p = &x;
```

A) `int`
B) `int&`
C) `int*`
D) `int**`

Question 3. Que fait l'opérateur `*` appliqué à un pointeur ?

A) Il multiplie la valeur pointée par 2
B) Il retourne l'adresse du pointeur
C) Il accède à la valeur stockée à l'adresse contenue dans le pointeur
D) Il libère la mémoire pointée

Question 4. Quelle est la bonne manière d'initialiser un pointeur qui ne pointe vers rien en C++ moderne ?

A) `int* p = 0;`
B) `int* p = NULL;`
C) `int* p = nullptr;`
D) `int* p;`

Question 5. Qu'affiche le programme suivant ?

```
int a = 5;
int* p = &a;
*p = 10;
std::cout << a;
```

A) 5
B) 10
C) L'adresse de `a`
D) Une erreur de compilation

Question 6. Comment libérer la mémoire allouée par `new int(42)` ?

A) `free(p);`
B) `delete[] p;`
C) `delete p;`
D) La mémoire est libérée automatiquement

Question 7. Comment libérer la mémoire allouée par `new int[10]` ?

A) `delete p;`
B) `delete[] p;`
C) `free(p);`
D) `delete[10] p;`

Question 8. Où sont stockées les variables locales d'une fonction ?

A) Sur le tas (heap)
B) Sur la pile (stack)
C) Dans un registre du processeur
D) Dans la mémoire statique

Question 9. Quelle affirmation est correcte concernant la pile (stack) et le tas (heap) ?

A) La pile est plus grande que le tas
B) Le tas est géré automatiquement par le compilateur
C) La pile a une taille limitée (quelques Mo) tandis que le tas peut utiliser plusieurs Go
D) Les deux ont la même taille

Question 10. Qu'est-ce qu'un pointeur dangling (pointeur pendu) ?

A) Un pointeur initialisé à `nullptr`
B) Un pointeur qui pointe vers une zone mémoire déjà libérée ou invalide
C) Un pointeur qui pointe vers le tas
D) Un pointeur de type `void*`

Question 11. Qu'affiche le programme suivant ?

```
int tab[3] = {10, 20, 30};
int* p = tab;
std::cout << *(p + 2);
```

A) 10
B) 20
C) 30
D) L'adresse de `tab[2]`

Question 12. Quelle est la différence entre `const int* p` et `int* const p` ?

A) Il n'y a aucune différence
B) `const int* p` : on ne peut pas modifier la valeur pointée ; `int* const p` : on ne peut pas modifier l'adresse stockée dans `p`
C) `const int* p` : on ne peut pas modifier l'adresse ; `int* const p` : on ne peut pas modifier la valeur pointée
D) Les deux empêchent toute modification

Question 13. Quel est l'affichage du programme suivant ?

```
int a = 10, b = 20;
int* p = &a;
p = &b;
*p = 30;
std::cout << a << " " << b;
```

A) `10 30`
B) `30 20`
C) `10 20`
D) `30 30`

Question 14. Quel problème présente le code suivant ?

```
void f() {
    int* p = new int(10);
    // fin de la fonction sans delete
}
```

A) Double libération
B) Fuite mémoire (memory leak)
C) Pointeur dangling
D) Aucun problème

Question 15. Quelle est la taille d'un pointeur (`int*`) sur une machine 64 bits ?

A) 4 octets
B) 8 octets
C) La même taille que `int`
D) Cela dépend du type pointé

Question 16. Que signifie l'expression `tab[i]` pour un tableau `tab` en termes d'arithmétique de pointeur ?

A) `&tab + i`
B) `*(tab + i)`
C) `*tab + i`
D) `tab * i`

Question 17. Quel est le type de retour de `malloc` en C ?

A) `int*`
B) `char*`
C) `void*`
D) `size_t`

Question 18. Quel problème présente le code suivant ?

```
int* p = new int(5);
delete p;
delete p;
```

A) Fuite mémoire
B) Double libération (double free) — comportement indéfini
C) Aucun problème

D) Erreur de compilation

Question 19. Quel est l'affichage du programme suivant ?

```
int a = 5, b = 10;
int* p = &a;
int* q = &b;
*p = *q;
std::cout << a << " " << b;
```

A) `10 10`
B) `5 5`
C) `5 10`
D) `10 5`

Question 20. Quel est l'affichage du programme suivant ?

```
int tab[] = {10, 20, 30, 40};
int* p = tab;
std::cout << *(p + 1) + *(p + 3);
```

A) 30
B) 50
C) 60
D) 70

Question 21. Que se passe-t-il dans le code suivant ?

```
int* createValue() {
    int a = 42;
    return &a;
}
int main() {
    int* p = createValue();
    std::cout << *p;
}
```

A) Affiche 42
B) Erreur de compilation
C) Comportement indéfini : `a` est détruite à la fin de `createValue`, le pointeur retourné est dangling
D) Affiche 0

Question 22. Considérons le code suivant. Quelle ligne provoque une erreur de compilation ?

```
int x = 10;
const int* const p = &x;
*p = 20;   // ligne A
p = nullptr; // ligne B
```

A) Seule la ligne A
B) Seule la ligne B
C) Les lignes A et B
D) Aucune ligne ne provoque d'erreur

Question 23. Soit le code suivant :

```
int tab[5] = {1, 2, 3, 4, 5};
int* p = tab + 2;
p[-1] = 10;
```

Quel est le contenu de `tab` après exécution ?

A) `{1, 2, 10, 4, 5}`
B) `{1, 10, 3, 4, 5}`
C) `{10, 2, 3, 4, 5}`
D) Comportement indéfini

Question 24. Quel est le problème avec le code suivant ?

```
int* tab = new int[100];
delete tab;
```

A) Aucun problème
B) Fuite mémoire
C) Comportement indéfini : il faut utiliser `delete[]` pour libérer un tableau alloué avec `new[]`
D) Erreur de compilation

Question 25. Quel est l'affichage du programme suivant ?

```
void swap(int* a, int* b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main() {
    int x = 3, y = 7;
    swap(&x, &y);
    std::cout << x << " " << y;
}
```

A) `3 7`
B) `7 3`
C) `7 7`
D) Erreur de compilation