

NOM :

Prénom :

## B23 - Introduction à la programmation

Durée : 1h30,  
Nombre de pages : 8  
Documents et calculatrices autorisés

Les exercices seront réalisés directement sur les sujets.

### Exercice 1 – Mystère (3 points)

On considère le programme en C suivant :

```
#include <stdio.h>

int mystere(int x, int y) {
    int z, w;
    z = 0;
    w = x;

    while(w<=y){
        z = z + w*w;
        w=w+1;
    }

    return z;
}

int main() {

    int i, j;
    printf("Entrer deux valeurs\n");
    scanf("%d %d", &i, &j);

    printf("le resultat de la fonction mystere est: %d\n", mystere(i,j));

    return 0;
}
```

Q 2.1) Quel est le résultat affiché par ce programme pour deux les valeurs saisies i = 2, j=3

Q 2.2) Quel est le résultat affiché par ce programme pour les deux valeurs saisies i = 3, j=6

Q 2.3) Que fait la fonction mystere?

## Exercice 2 – Fonctions (2 points)

Q 2.1) Écrire une fonction qui permet de calculer la somme de diagonal d'une matrice carrée *Mat* de taille  $m \times m$ .

Par exemple, pour la matrice  $M$  :  $M = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 5 \\ 4 & 3 & 7 \end{bmatrix}$

$$\text{SommeDiag}(M) = 1 + 9 + 7 = 17$$

```
int SommeDiag (int Mat[m][m]) {
```

```
}
```

## Exercice 3 – Structures (7 points)

L'objectif de cet exercice est de gérer la collection de Pokémon© d'un dresseur Pokémon.

La structure permettant de créer un Pokémon est définie comme :

```
struct PokemonSt{
    char nom[50] ; //le nom du Pokémon
    int niveau ;    //le niveau du Pokémon
    int PV;         //le nombre de points de vie
    int PC;         //points de combat : le nombre de points de vie retirés à l'adversaire lors d'une attaque
}
typedef struct PokemonSt Pokemon ;
```

Q 3.1) Donnez l'implémentation de la procédure `void printPokemon(Pokemon p)` qui affiche toutes les caractéristiques d'un Pokémon *p*.

```
void printPokemon (Pokemon p){
```

```
}
```

Q 3.2) Donnez l'implémentation de la fonction *Pokemon evolution* (*Pokemon p*) qui fait évoluer un Pokémon (incrémente son niveau, augmente son nombre de points de vie de 20 et sa force de 10).

```
Pokemon evolution(Pokemon p){
```

```
}
```

Q 3.3) Donnez l'implémentation de la fonction *Pokemon attaque*(*Pokemon p1, Pokemon p2*) simule une attaque du Pokémon p1 sur le Pokémon p2. Il faudra mettre à jour les points de vie du Pokémon attaqué (N.B. si un Pokémon a moins de points de vie que la force de son adversaire, son nombre de points de vie sera mis à 0).

```
Pokemon attaque(Pokemon p1, Pokemon p2) {
```

```
}
```

Un dresseur Pokémon a un nombre *N* de Pokémon. Il peut les stocker dans un tableau de Pokémon. Soit :

```
#define MAX_POKEMON 9999  
Pokemon Tab[MAX_POKEMON] ;
```

Q 3.4) Donnez l'implémentation de la fonction *int nbPokemonsVivants(Pokemon Tab*[MAX\_POKEMON], *int N*) qui renvoie le nombre de Pokemons vivants du dresseur.

```
int nbPokemonsVivants(Pokemon Tab[MAX_POKEMON], int N){
```

```
}
```

Q 3.5) Donnez l'implémentation de la fonction `Pokemon pirePokemon(Pokemon Tab[MAX_POKEMON], int N)` qui renvoie le Pokemon du dresseur qui a la force la plus faible (on supposera que les Pokémons ont tous une force différente).

```
Pokemon pirePokemon(Pokemon Tab[MAX_POKEMON], int N){
```

```
}
```

Q 3.6) Donnez l'implémentation de la fonction `void supprimePirePokemon(Pokemon Tab[MAX_POKEMON], int *N)` qui supprime le Pokemon du dresseur qui est le plus faible et qui renvoie le nouveau nombre de Pokémons.

```
void supprimePirePokemon(Pokemon Tab[MAX_POKEMON], int *N){
```

```
}
```

Q 3.7) Écrire un programme principal `int main()` qui permet d'appeler toutes ces précédentes fonctions et procédures, en n'oubliant pas de déclarer les variables nécessaires.

```

int main(){

// déclaration et initialisation d'un tableau de Meubles
Pokemon Tab [MAX_POKEMON];
int N=3 ;
Tab [0].nom = "Pikachu" ;    Tab[0].niveau = 3 ;
Tab [0].PV = 15 ;            Tab [0].PC = 26;

Tab [1].nom = "Miaouss" ;    Tab [1].niveau = 1 ;
Tab [1].PV = 8 ;            Tab [1].PC = 12 ;

Tab [2].nom = "Rondoudou" ; Tab [2]. niveau = 2 ;
Tab [2].PV = 23;            Tab [2].PC = 5;

// Afficher le Pokémon Rondoudou


// Afficher tous les élément du tableau


// faites évoluer le Miaouss dans le tableau


// simulez une attaque de Miaouss sur Pikachu


// affichez le nombre de Pokémon encore vivants


// supprimez le Pokémon la plus faible


    return 0 ;
}

```

#### Exercice 4 – Tableaux (3 points)

Q 4.1) Ecrire une fonction *float Moyenne* (int Notes[], int N) qui prend un tableau *Note* de *N* notes d'une classe et renvoie la moyenne générale de notes dans le tableau.

```

float Moyenne (int Notes[], int N) {

```

```
}
```

Q 4.2) Ecrire une fonction *float Pourcentage* (*int Notes[]*, *int N*) qui renvoie le pourcentage de notes supérieures à la moyenne générale de notes dans le tableau.

```
float Pourcentage (int Notes[], int N) {
```

```
}
```

Q 4.3) Ecrire une fonction *int valide* (*int Notes[]*, *int N*) qui renvoie le nombre de notes supérieurs à la moyenne de validation (10).

```
int valide (int Notes [ ], int N) {
```

```
}
```

### Exercice 5 – Chaînes de caractères (5 points)

Q 5.1) La distance de Hamming entre deux mots est une notion utilisée dans de nombreux domaines (télécommunications, traitement du signal, . . . ). Elle est définie, pour deux mots de même longueur, comme le nombre de positions où les deux mots ont un caractère différent.

Écrire une fonction *int Hamming* ( *char S1 []*, *char S2 []* ) qui calcule la distance de Hamming entre deux mots *S1* et *S2* lorsqu'ils ont la même longueur, et qui renvoie -1 sinon.

Par exemple : `hamming("aaba", "aaha")` renvoie 1,  
`hamming("poire", "pomme")` renvoie 2 et

hamming("stylo", "bouteille") renvoie -1.

```
int Hamming ( char S1 [], char S2 [] ) {
```

```
}
```

Q 5.2) Écrire une procédure *void suppression(char c, char S[])* qui supprime la première occurrence d'un caractère *c* s'il apparaît dans *S*. Si *c* n'apparaît pas dans *S*, la chaîne reste inchangée. Cette fonction ne doit pas utiliser de tableau intermédiaire.

Par exemple : *suppression("a", "baldaquin")* permet de modifier la chaîne en "bldaquin"  
*suppression("d", "fleur")* permet de garder la chaîne "fleur" inchangée.

```
void suppression ( char c, char S[] ) {
```

```
}
```

Q 5.3) Écrire une fonction *cherche* qui cherche un caractère *c* dans une chaîne de caractères *S* et renvoie 1 si *c* apparaît dans *S*, et 0 sinon.

Par exemple : *cherche("a", "cheval")* renvoie 1  
*cherche("a", "école")* renvoie 0.

```
int cherche ( char c, char S[] ) {
```

```
}
```

Q 5.4) Écrire une fonction *int cherche2(char c, char S [])* qui cherche un caractère *c* dans une chaîne de caractères *S* et renvoie la position de la première occurrence du caractère dans la chaîne et si le caractère n'est pas présent la fonction renvoie -1.

Par exemple : `cherche2("a","ecole")` doit renvoie -1  
`Cherche2("a","cheval")` doit renvoie 4.

```
int cherche2(char c, char S []) {
```

```
}
```