# LIFAP1 – CC mi-parcours – Séquence 5

**Contrôle Continu** (Durée totale : 1h)
Vendredi 5 novembre 2021

NOM :

……

PRENOM :

……

Numéro Etudiant :

……

***Recommandations :*** *Les documents, calculatrice, téléphone portable sont interdits. La qualité de l'écriture et de la présentation seront prises en compte dans la note finale. Vous veillerez à* ***respecter*** *les notations et les règles d'écriture des algorithmes vues en cours et en TD. Un soin tout particulier devra être apporté à l'écriture des entêtes des différents sous-programmes.*

## Partie A – Algorithmique

**Évolution d'une somme placée en banque.**

On place à la banque un capital de départ de 500 €. Chaque année le capital augmente avec un taux d'intérêt de 2 % et chaque année on rajoute 100 €.

La valeur du capital à l'année n est obtenu par la suite $u_n$ définie par :

$u_0 = 500$

$u_{n+1} = 1{,}02 \times u_n + 100$

1. Écrire l'algorithme d'un **sous-programme** `tab_suite` qui remplit un tableau de taille `MAX` (`MAX` constante définie au préalable à la valeur 100) avec les `MAX` premiers termes de la suite.

```
Procédure tab_suite( T : tableau [MAX] de réels)
Précondition : aucune
Donnée : aucune
Données / résultats : T
Description : remplit T avec les MAX premiers termes de la suite
Variables locales : i : entier
Début
        T[0] ← 500
        Pour i allant de 1 à MAX-1 par pas de 1 faire
                T[i] ← 1.02 * T[i-1] +100
        Fin pour
Fin
```

2. Écrire l'algorithme d'un **sous-programme** `capital_superieur` qui, à partir du tableau précédent retourne le nombre d'années nécessaires pour dépasser une certaine valeur de `capital` passé en paramètre. Si le montant souhaité reste supérieur à la valeur de la dernière case du tableau, la fonction renverra `-1`.

```
Fonction  tab_suite( T : tableau [MAX] de réels, capital : réel) : entier
Précondition : aucune
Donnée : aucune
Données / résultats : T
Résultat : entier
Description : retourne le nombre d'années nécessaire pour dépasser un certain capital
Variables locales : i, nb : entier
Début
        i ← 0
        tant que (T[i]<=val et  i<MAX)
                i← i+1
        fin tant que
        si (val>T[i])
                i← -1
        fin si
        retourner i
Fin
```

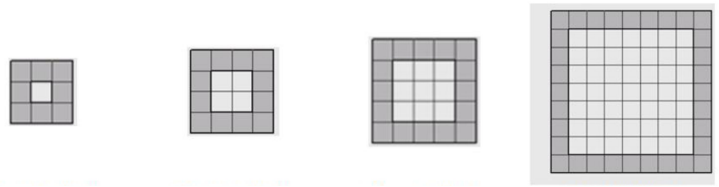
3. Écrire l'algorithme du **programme principal** qui demande à l'utilisateur le capital à atteindre et lui affiche le nombre d'années nécessaires pour obtenir ce capital. On passera par la construction du tableau contenant toutes les valeurs de la suite.

```
Début
        temps : entier
        interet : tableau [MAX] de réels
        capital : réel
        saisir(capital)
        remplit_tab_suite(interet)
        temps ← capital_superieur(interet, capital)
        afficher ("le capital devient superieur au bout de ",temps," annees")
Fin
```

# Partie B – Langage C/C++

**Les carrés bordés**

On veut déterminer le nombre de carrés blancs et le nombre de carrés gris dans des configurations de carré différentes, tels que ceux-ci-dessous.



n = 3 n = 4 n = 5 n = 9

Pour un carré de taille n, il y a 4(n-1) carrés noirs et (n - 2)² carrés blancs ; la somme des deux doit être égale à n².

1- Écrire en langage C/C++ un sous-programme `compte_blancs_noirs` qui calcule et "retourne" le nombre de carrés blancs et le nombre de carrés noirs en fonction de la taille du carré, `n`, passée en paramètre.

```
void noir_et_blanc(int N, int &noirs, int &blancs)
{
   noirs = 4 * ( N-1);
   blancs = pow(N-2,2);
}
```

2- Définir en C/C++ une constante `MAX` ayant comme valeur 13.

```
const int MAX = 13 ;
```

3- Écrire en langage C/C++ un sous-programme qui remplit un tableau 2D de taille `2` lignes et `MAX` colonnes contenant dans la première ligne le nombre de carrés noirs et dans la seconde le nombre de carrés blancs.

```
void remplit_tab_2D(int Tab[2][MAX], int N)
{
   int i, n,b;
   for (i=2; i<=MAX+2; i++)
   {
      noir_et_blanc(i,n,b);
      Tab[0][i-2] = n;
      Tab[1][i-2] = b;

   }
}
```

4- Écrire en langage C/C++ une **fonction booléenne** `noirs_plus_blancs` qui retourne vrai si pour une valeur `n` passée en paramètre la somme du nombre de carré noirs et de carrés blancs est bien égale à $n^2$, faux sinon. On utilisera impérativement le tableau construit à la question précédente.

```
bool noirs_plus_blancs (int Tab[2][MAX], int n)
{
        return ((Tab[0][n-2] + Tab[1][n-2] ) == n*n) ;
}
```

5- Écrire en langage C/C++ une **fonction** `saisie_taille_carre` qui demande à l'utilisateur de choisir une valeur comprise entre 3 et 15 (inclus) et la retourne au programme principal. La saisie sera recommencée tant que la valeur ne satisfait pas les contraintes.

```
int saisie_taille_carre ()
{
        int n ;
        do
        {
                cout<< "donnez un entier compris entre 3 et 15"<<endl ;
                cin >> n ;
        } while (n<3 || n>MAX+2);
        return n ;
}
```

6- Écrire en langage C/C++ le programme principal qui permet de construire le tableau contenant pour toutes les valeurs de n comprises entre 3 et 15 le nombre de carrés noirs et le nombre de carrés blancs, puis de saisir une valeur `n` comprise entre 3 et 15 et de vérifier que la somme des carrés blancs et des carrés noirs est bien égale à $n^2$.

```
int main (void)
{
    int T[2][MAX];
    int n = saisie_taille_carre();
    remplit_tab_2D(T,MAX);
    cout<<noirs_plus_blancs(T,n);
  return 0 ;
}
```