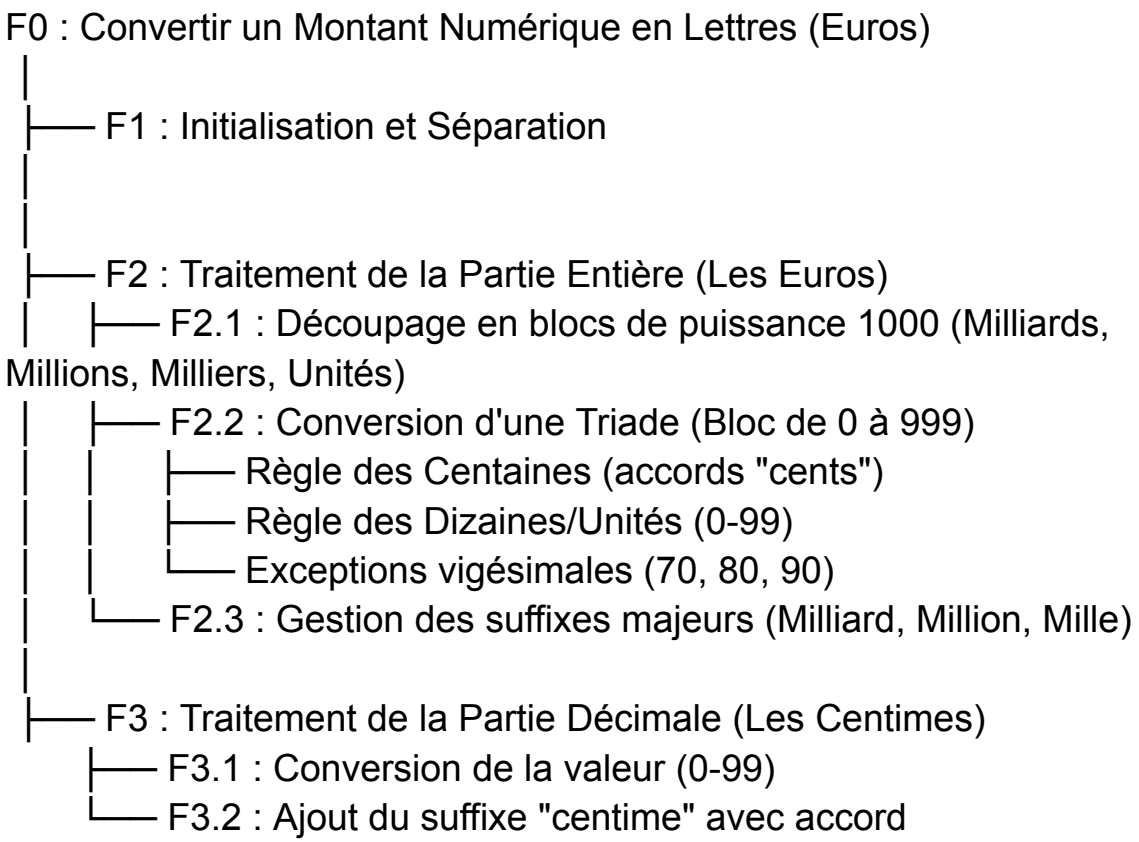
Analyse : Transcription nombre -> montant Euro

BRAGA Paulo | LE DÛ Millan
BUT 2 | Groupe 3.1

Analyse Hiérarchique Fonctionnelle :



FS 0	Convertir un Montant Numérique en Lettres (Euros)

VA

Orchestrer la transformation d'un `double` en phrase complète

Schéma fonctionnel

Entrées : Nombre réel `montant` (ex: 1250.50)

Sorties : Chaîne de caractères complète

Logigramme OU Algo

1) Si montant = 0, renvoyer "zéro euro"
2) Exécuter **F1** (Séparation)
3) Exécuter **F2** (Partie Entière).
4) Ajouter le mot "euro" (ou "euros" si Entier > 1)
5) Si décimale > 0, ajouter "et", puis exécuter **F3**
6) Exécuter **F4** (Retour chaîne)

FS 1.0	Initialisation et Séparation

VA

Permet de préparer les données brutes

Schéma fonctionnel

Entrées : `montant` (double)

Sorties : `long long entier`, `int centimes`

Logigramme OU Algo

`entier` = partie entière du montant

`centimes` = arrondi à 2 chiffres après la virgule : `round((montant - entier) * 100)`

FS 2.0	Traitement Partie Entière

VA

Convertir le nombre principal

Schéma fonctionnel

Entrées : `long long entier`.

Logigramme OU Algo

1) Si montant = 0, renvoyer "zéro euro"

2) Exécuter **F1** (Séparation)

3) Exécuter **F2** (Partie Entière).

4) Ajouter le mot "euro" (ou "euros" si Entier > 1)

5) Si décimale > 0, ajouter "et", puis exécuter **F3**

6) Exécuter **F4** (Retour chaîne)

FS 3.0	Traitement Partie Décimale

VA

Gérer les centimes

Schéma fonctionnel

Entrées : `int centimes`

Logigramme OU Algo

Utilise la même logique de conversion que pour un nombre entier simple (0-99), mais ajoute le mot "centime(s)" à la fin

FS 2.1	Découpage en Triades

VA

Isoler les groupes de 3 chiffres pour traitement

Schéma fonctionnel

Entrées : Le nombre entier total

Logigramme OU Algo

Tant que `nombre >= 1 000 000 000` -> Traiter Milliards

Tant que `nombre >= 1 000 000` -> Traiter Millions

Tant que `nombre >= 1 000` -> Traiter Milliers

Reste -> Traiter Unités

FS 2.2	Conversion d'une Triade (0-999)

VA

Convertir n'importe quel nombre entre 0 et 999 en texte

Schéma fonctionnel

Entrées : `int n` (le nombre), `bool est_fin_gros_nombre` (pour l'accord de "cent")

Logigramme OU Algo

Centaines : Si `n >= 100` : Traduire le chiffre des centaines + "cent"

Reste (0-99) :

- **0-19 :** Dictionnaire direct (un, deux... dix, onze... dix-neuf)
- **20-69 :** Dictionnaire Dizaine (vingt... soixante) + tiret + Unité
- **70-79 :** Mot "soixante" + convertir (Reste - 60) comme un 10-19
- **80-89 :** Mot "quatre-vingt"
- **90-99 :** Mot "quatre-vingt" + convertir (Reste - 80) comme un 10-19

Liaisons "et" : Pour 21, 31, 41, 51, 61, 71 -> Ajouter "et"

FS 2.3	Gestion des Suffixes Majeurs

VA

Déterminer le suffixe correct à apposer après un bloc de 3 chiffres, en gérant le pluriel et l'élision

Schéma fonctionnel

Entrées : `int valeur_bloc` : La valeur du bloc de 3 chiffres qui vient d'être traité (ex: 1, 200, 0)

`enum echelle` : Indique si on traite des MILLIERS (10^3), MILLIONS (10^6) ou MILLIARDS (10^9)

Logigramme OU Algo

```
SI Valeur == 0 ALORS
   Retourner CHAINE_VIDE
FIN SI

SELON Type :
   CAS MILLIER :
      Retourner "mille"  (Toujours invariable)

   CAS MILLION :
      SI Valeur > 1 ALORS Retourner "millions"
      SINON Retourner "million"

   CAS MILLIARD :
      SI Valeur > 1 ALORS Retourner "milliards"
      SINON Retourner "milliard"
FIN SELON
```

FS 3.1	Conversion de la valeur (0-99)

VA

Convertir le nombre de centimes en texte

Schéma fonctionnel

Entrées : N (entier entre 0 et 99).

Logigramme OU Algo

1) Si N = 0 : Terminer

2) Si N < 20 :
   Utiliser le dictionnaire des unités (ex: "seize")

3) Si N >= 20 :
   Trouver la dizaine (ex: "vingt", "soixante-dix"...)
   Ajouter la liaison ("-" ou "et")
   Ajouter l'unité restante

FS 3.2	Ajout du suffixe “centime” avec accord

VA

Mettre le mot centime au pluriel si nécessaire

Schéma fonctionnel

Entrées : N (Nombre de centimes)

Logigramme OU Algo

1) Si N = 0 : Terminer

2) Si N > 1 :
   Écrire "centimes"

3) Sinon (donc N = 1) :
   Écrire "centime"

Code source :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>


const char *UNITES[] = {"", "un", "deux", "trois", "quatre", "cinq", "six", "sept", "huit",
"neuf"};
const char *DIX_A_DIXNEUF[] = {"dix", "onze", "douze", "treize", "quatorze",
"quinze", "seize", "dix-sept", "dix-huit", "dix-neuf"};
const char *DIZAINES[] = {"", "", "vingt", "trente", "quarante", "cinquante", "soixante",
"soixante", "quatre-vingt", "quatre-vingt"};


char buffer[1024];

// Prototypes
void ajouter_mot(const char *mot);
void convertir_0_99(int n);
void convertir_bloc_999(int n, int est_fin_du_nombre);
void convertir_complet(double montant);



void ajouter_mot(const char *mot) {
    if (buffer[0] != '\0') {
        strcat(buffer, " ");
    }
    strcat(buffer, mot);
}

// Convertir de 0 à 99
void convertir_0_99(int n) {
    if (n == 0) return;

    // Cas 1 : 1 à 9
    if (n < 10) {
        ajouter_mot(UNITES[n]);
    }
    // Cas 2 : 10 à 19
    else if (n >= 10 && n < 20) {
```

```
        ajouter_mot(DIX_A_DIXNEUF[n - 10]);
    }
    // Cas 3 : 20 à 99
    else {
        int dizaine = n / 10;
        int unite = n % 10;

        // Gestion spéciale pour 70-79 et 90-99
        if (dizaine == 7 || dizaine == 9) {
            dizaine -= 1;
            unite += 10;
        }

        char mot_dizaine[50];
        strcpy(mot_dizaine, DIZAINES[dizaine]);

        // Règle du "s" pour 80

        if (n == 80) {
            strcat(mot_dizaine, "s");
        }

        ajouter_mot(mot_dizaine);

        // Gestion du "et un"
        // Avec 81 et 91 qui ne prennent pas de "et"
        if (n % 10 == 1 && n != 81 && n != 91) {
            strcat(buffer, "-et");
        } else if (unite < 20 && unite > 0) {
             strcat(buffer, "-");
             // On enlève le dernier espace ajouté par ajouter_mot implicitement s'il y en
avait

             buffer[strlen(buffer)-1] = '\0'; // Supprimer l'espace du prochain mot
             strcat(buffer, "-");
        }

        // Appel récursif pour l'unité
        if (unite < 10) {
            if (unite > 0) ajouter_mot(UNITES[unite]);
        } else {
            // Gestion des cas de 70 et 90)
            ajouter_mot(DIX_A_DIXNEUF[unite - 10]);
        }
```

```
    }
}

// Convertir un bloc de 3 chiffres (0-999)
void convertir_bloc_999(int n, int est_fin_du_nombre) {
    if (n >= 100) {
        int centaines = n / 100;
        int reste = n % 100;

        if (centaines > 1) {
            ajouter_mot(UNITES[centaines]);
        }

        // Règle du pluriel de "cent"
        if (centaines > 1 && reste == 0 && est_fin_du_nombre) {
            ajouter_mot("cents");
        } else {
            ajouter_mot("cent");
        }

        convertir_0_99(reste);
    } else {
        convertir_0_99(n);
    }
}

// Le convertisseur à appeler
void convertir_complet(double montant) {
    // Reset buffer
    buffer[0] = '\0';

    if (montant == 0) {
        printf("zéro euro\n");
        return;
    }

    // Séparation
    long long entier = (long long)montant;
    int centimes = (int)round((montant - entier) * 100);

    if (entier == 0) {
        ajouter_mot("zéro");
    }
```

```
    // Découpage
    long long r;

    // Milliards
    if (entier >= 1000000000) {
        r = entier / 1000000000;
        convertir_bloc_999((int)r, 0);
        ajouter_mot(r > 1 ? "milliards" : "milliard");
        entier %= 1000000000;
    }

    // Millions
    if (entier >= 1000000) {
        r = entier / 1000000;
        convertir_bloc_999((int)r, 0);
        ajouter_mot(r > 1 ? "millions" : "million");
        entier %= 1000000;
    }

    // Milliers
    if (entier >= 1000) {
        r = entier / 1000;

        if (r > 1) {
            convertir_bloc_999((int)r, 0);
        }
        ajouter_mot("mille");
    }

    // Unités restantes
    if (entier > 0) {
        convertir_bloc_999((int)entier, 1);
    }

    // Ajout de euros ou euro
    ajouter_mot(((long long)montant) > 1 || ((long long)montant) == 0 ? "euros" :
"euro");

    // Décimales
    if (centimes > 0) {
        ajouter_mot("et");
        convertir_bloc_999(centimes, 1);
        ajouter_mot(centimes > 1 ? "centimes" : "centime");
    }
```

```
    // Affichage final
    printf("%s\n",buffer);
}


int main() {

    convertir_complet(0.0);
    convertir_complet(5.0);
    convertir_complet(21.0);
    convertir_complet(80.0);
    convertir_complet(82.0);
    convertir_complet(200.0);
    convertir_complet(205.0);
    convertir_complet(1234.56);
    convertir_complet(3000.00);
    convertir_complet(1000000);
    convertir_complet(2000000);
    convertir_complet(1999.99);
    convertir_complet(91.0);
    convertir_complet(2000000001);

    return 0;
}
```