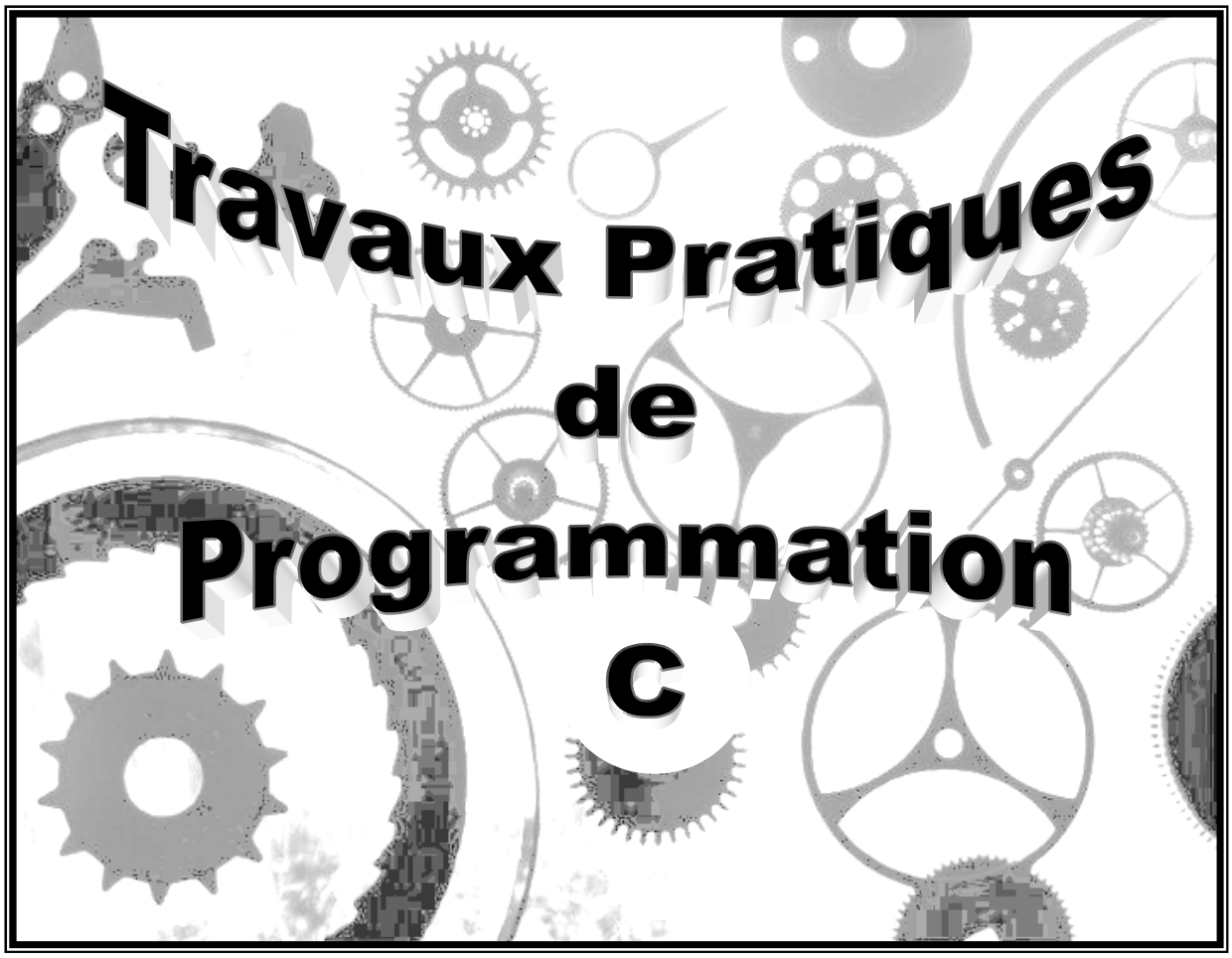
UNIVERSITÉ DE POITIERS

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
6, allée Jean Monnet



Nom :

Groupe :

2012-2013

Liste des TP

TP N°1 : PRISE EN MAIN DE L'ENVIRONNEMENT DE DEVELOPPEMENT DEV-C++
SEQUENCES – ALTERNATIVES 1

1. Introduction 1
2. Fonctionnement en réseau 1
3. Manipulation de répertoires et de fichiers 1
4. Création d'un projet – élaboration d'un programme 2
   4.1. Création du projet 2
   4.2. Déclaration de variables 3
   4.3. Compilation - exécution 3
      4.3.1. Vérification de la syntaxe 3
      4.3.2. Compilation + exécution 3
5. Syntaxe de printf() 4
   5.1. Ecriture de message simple 4
   5.2. Ecriture de message et de contenus de varaiables 4
6. Syntaxe de scanf() : lecture de varaibles de type float et int 6
7. Structure alternative 7
   7.1. Exemple 7
   7.2. Résolution de l'équation du premier degré 7
   7.3. Développement 8

Travaux supplémentaires

8. Résolution de l'équation du 2e degré 10
9. Validation d'une date 15

TP N°2 : ITERATIONS 19
1. Méthodologie d'analyse d'une structure itérative 19
2. Ecriture en langage C de diverses formes d'itérations 19
2.1. Exercice n°1 : somme des N premiers entiers 19
2.2. Exercice n°2 : somme des N premiers entiers carrés 20
2.3. Exercice n°3 : détection d'une touche clavier 20
3. Calcul de la puissance entière d'un nombre réel 22

Travaux supplémentaires

3. Calcul du pgcd de deux nombres entiers 25
3.1. Technique des soustractions successives 25
3.2. Algorithme d'Euclide : technique des divisions successives 27
4. développement limité de la fonction exp 29

TP N°3 : TABLEAU DE NOMBRES 33

1. Manipulations élémentaires, édition 33
2. Opérations statistiques sur les valeurs d'un tableau de nombres 36

Travaux supplémentaires

3. Extraction de la valeur minimale et maximale 38
4. Triangle de Pascal – Tableau à 2 dimensions 39

TP N°4 : FONCTIONS EN C 41

1. Introduction 43
2. Exemple : fonction difference 44
3. Exercice de base de développement de fonctions 46
3.1. Fonction min 46
3.2. Fonction max 47
3.3. Fonction valeur_absolue 48
3.4. Fonction majuscule 49
3.5. Fonction nb_chiffres 50
4. Calcul des valeurs de gain et de phase d'un filtre passe-bas 51
4.1. Ecriture de la fonction gain_passe_bas 52
4.2. Ecriture de la fonction phase_passe_bas 52
4.3. Ecriture du programme de construction des tableaux Freq, Gains et Phases 53

Travaux supplémentaires

5. Calcul du pgcd 56

TP N°5: POINTEURS ET PROCEDURES 59

1. Pointeurs 59
2. Ecriture d'une procédure permute 60
3. Développement d'un programme de résolution d'une équation du premier degré découpé en modules 62

Travaux supplémentaires

4. Développement d'un sous-programme puissance 65
5. Développement d'une procédure de développement limité de la fonction exp(*x*) 67

TP N°6 : SOUS-PROGRAMMES : MANIPULATION DE TABLEAUX 71

1. Pointeurs et tableaux à une dimension 71
2. Procédure de lecture des valeurs significatives d'un tableau de nombres 72
3. Procédure d'affichage d'un tableau de nombres 73
4. Fonction min_ou_max 74
5. Fonction select_positifs 76

Travaux supplémentaires

6. Conception d'un jeu de réflexion – jeu des permutations 78
   - 6.1. Principe du jeu 78
   - 6.2. Programmation 81
     - 6.2.1. Fonction lit_entier_dans_intervalle 81
     - 6.2.2. Procédure affiche_liste 81
     - 6.2.3. Procédure melange 82
     - 6.2.4. Procédure choix_1er_element 83
     - 6.2.5. Fonction test_liste_ordonnee 85

TP N°7 : PROGRAMME DE CRYPTAGE DE CHAINES DE CARACTERES : LE CODE DE CESAR 87

1. Le code de César 87
2. Principe du codage d'une lettre majuscule (fonction decalage) 87
3. Codage d'une chaîne de caractères 91
4. Détermination du code d'un message codé 92
   - 4.1. Décalage des caractères à gauche ou à droite d'un message 92
   - 4.2. Recherche du code d'un message codé (procédure recherche_code) 94
   - 4.3. Décodage direct d'un message avec le code connu 97

Sujet supplémentaire de synthèse : Programme de tri à bulle 99
1. Première étape : recherche de la valeur minimale 99
2. Deuxième étape : tri à bulle par ordre croissant 101

Sujet supplémentaire de synthèse : Jeu puissance 4 103
1. Introduction 103
2. Codage de la position des pions 103
3. Structure du programme 104
3.1. Les données du programme principal 104
3.2. Le corps exécutable du programme principal 104
3.2.1. Procédure vide_grille 106
3.2.2. Procédure initialise 107
3.2.3. Procédure affiche_grille 107
3.2.4. Fonction depot_pion_colonne 109
3.2.5. Procédure remplissage_grille 110
3.2.6. Fonction recherche_ligne_4_pions 111
3.2.7. Edition des résultats 114
4. Résultat souhaité d'exécution lors du déroulement d'une partie 114

ANNEXES

ENVIRONNEMENT DE DEVELOPPEMENT INTEGRE DEV-C++

1. Introduction 1
2. Lancement de Dev-C++ à partir de Windows : 1
3. Présentation de l'environnement Dev-C++ : 2
3.1. Les différentes icônes directement accessibles 3
3.2. Les commandes d'exécution en mode pas à pas 4
3.3. Résumé des actions de compilation 5

TABLE DES CODES ASCII STANDARD 7

PRISE EN MAIN DE L'ENVIRONNEMENT DE DEVELOPPEMENT DEV-C++

SEQUENCES - ALTERNATIVES

Thèmes abordés : *instructions* **`printf`**, **`scanf`**, *(manuel* **Eléments de base du langage C**, *p.4-8) , structures alternatives (manuel* **Eléments de base du langage C**, *p.12-13)*

1. Introduction

Ce premier T.P. est destiné à vous familiariser avec le système informatique en général et à l'environnement intégré Dev-C++ en particulier.

Objectif : chargement, sauvegarde de fichiers dans les bons répertoires; gestion de projet ; instructions de lecture **`scanf`** et d'écriture **`printf`**; création d'un programme documenté avec manipulation de variables de type simple. Les structures alternatives.

2. Fonctionnement en réseau

Un serveur fonctionnant sous WINDOWS XP gère les accès aux différents logiciels et les différents répertoires de tous les utilisateurs.

Lors de l'ouverture d'une session, il vous faudra :

1) Entrer votre nom d'utilisateur.

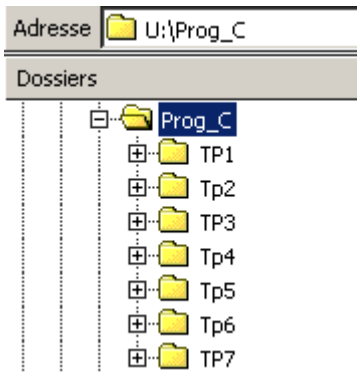
2) Entrer un mot de passe : il s'agit de votre mot de passe confidentiel.

Par la suite, tous vos travaux seront sauvegardés dans un répertoire physique géré par le serveur et affecté à l'unité logique **`U:>`**.

3. Manipulation de répertoires et de fichiers

Visualiser les sous-répertoires présents dans le répertoire **`U:\Prog_c`** et leurs contenus.
Par l'intermédiaire du gestionnaire de fichiers, créer les répertoires :

- **`U:\Prog_c\TP2`**,
- **`U:\Prog_c\TP3`**,
- **`U:\Prog_c\TP4`**,
- **`U:\Prog_c\TP5`**,
- **`U:\Prog_c\TP6`**,
- **`U:\Prog_c\TP7`**.



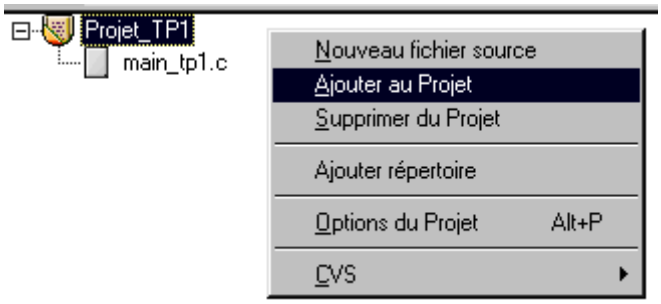
Les différents répertoires ainsi créés permettront de contenir l'ensemble des fichiers créés par l'environnement de développement intégré lors de l'élaboration de projets au cours des différents TP (répertoire **`U:\Prog_c\TP1`** pour le premier TP).

4. Création d'un projet – élaboration d'un programme

4.1. Création du projet

✓ Consulter l'annexe "*Environnement de développement intégré DEV-C++*" p.1 pour créer un fichier projet de nom **`Projet_TP1.dev`** dans votre répertoire **`U:\Prog_c\TP1`**.

✓ Fermer le fichier **`main.c`** sans l'enregistrer et dans l'onglet Projet, **ajouter** au projet **`Projet_TP1`** le fichier **`main_ecriture_lecture.c`** (Clic droit sur l'icône projet et **Ajouter au Projet**) On a donc un projet constitué d'un seul **fichier source** qui contient la fonction **`main`** et dont le listing est fourni **p 2**.



✓ Compléter le bloc d'identification constitué de lignes de commentaires (insérées entre **`/*`** et **`*/`**). Mettre à jour la date (**Edition** / **Insérer** / **Date/Heure**).




```
/* Nom de fichier disque : main_ecriture_lecture.c
   Projet : Projet_TP1.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : Premier programme : Instructions de lecture
                 (au clavier) et d'écriture
   Date :
*/

#include <stdio.h>
#include <stdlib.h>

int main()
{
  // Déclaration des variables du programme principal

  // Corps exécutable du programme principal

  // Edition

   system("PAUSE"); /* instruction permettant d'exécuter des
                       commandes en mode console (ici
                       pause d'exécution)   */
   return 0;
}
```

Bloc d'identification

✓ Consulter le contenu de votre répertoire **`U:\Prog_c\TP1`** : vous devez avoir deux fichiers : **`Projet_TP1.dev`** (fichier projet de **dév**eloppement) et **`main_tp1.c`**.

4.2. Déclaration de variables

En vous aidant du fascicule Eléléments de base du langage C (p.1 et 2), déclarer deux variables **`A`** et **`i`** de type **Réel** et **Entier**.

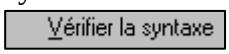
Dans le corps exécutable du programme, traduire en C les instructions d'affectation écrites en langage algorithmique :

i ← 1539

A ← 0.00001*i

4.3. Compilation – exécution (Menu Exécuter)

4.3.1. Vérification de la syntaxe

✓ Vérifier la syntaxe du texte écrit dans le fichier source **`main_tp1.c`** en activant la commande 

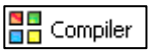
: **F1**.

L'analyseur syntaxique du compilateur permet d'indiquer d'éventuelles erreurs de syntaxe, mais aucun fichier binaire n'est généré.

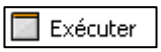
Ces messages doivent en principe permettre de déterminer rapidement l'erreur. Cependant, dans le cas où trop d'erreurs seraient présentes dans le texte source, les indications de la fenêtre des messages d'erreur peuvent vite devenir incompréhensibles. C'est pourquoi en adoptera la règle suivante :

En cours de saisie d'un texte source, vérifier la syntaxe toutes les 3 à 4 lignes (commande F1). Cette vérification s'accompagne automatiquement de la sauvegarde du fichier source

4.3.2. Compilation et exécution

Pour compiler le programme générant le code binaire, utiliser la commande 

.

Exécuter le programme en utilisant la commande 

.

Quel message obtenez-vous en activant cette commande ?

…….

Quelle intruction du programme est-elle à l'origine de ce message ?

……

Remarque : le fichier **`Projet_TP1.exe`** a été créé dans votre répertoire.

5. Syntaxe de printf () (Cf annexe Elements du langage C page 4)

5.1. Ecriture de message simple

La traduction en C de l'instruction :

Ecrire ("Premier programme ")

s'écrit :

```
printf ("Premier programme ") ;
```

Dans la partie édition du programme du fichier **`main_ecriture_lecture.c`** , faire la saisie de cette instruction puis exécuter le programme.

Modifier l'instruction en tapant :

```
printf ("Premier programme \n") ;
```

Que permet l'ajout du caractère **`\n`** dans la chaîne de caractères à afficher ?

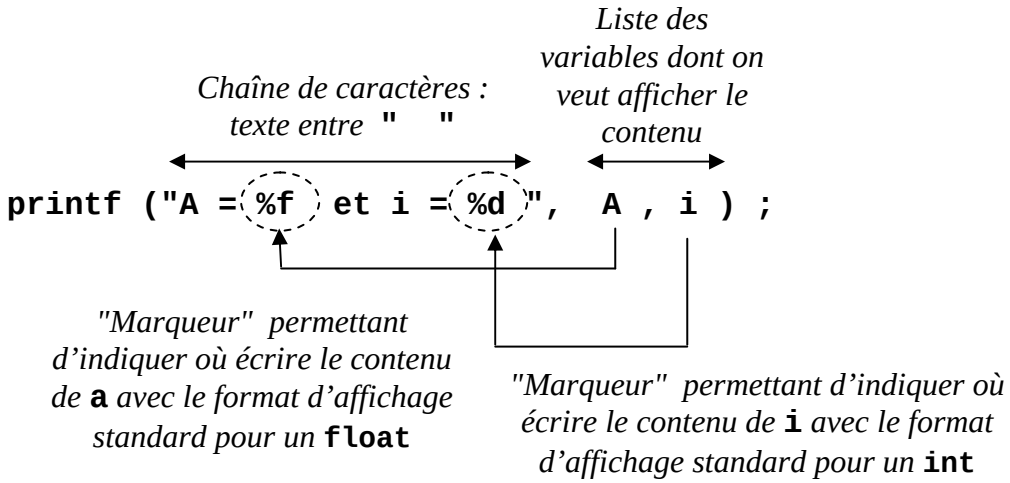
……

5.2. Ecriture de message et de contenu de variables

Soit à traduire en langage C l'instruction suivante écrite en langage algorithmique :

Ecrire ("A = ", **A** , " et i = ", **i**) où **A** est de type **Réel** et **i** de type **Entier**.

La traduction en C permettant l'affichage standard des contenus de **A** et **i** s'écrit :



Si **`A`** et **`i`** contiennent respectivement **0.001539** et **1539**, le résultat à l'écran sera :

```
A = 0.015390 et i = 1539
```

✓ Ecrire cette instruction et tester la en exécutant le programme.

Rappel : format d'affichage standard (voir fascicule Eléments du langage C p.5)

Déclaration	Syntaxe	Format externe
`int i ;`	`printf("%d",i) ;`	Décimal
`int i ;`	`printf("%x et %X",i,i) ;`	Héxadécimal
`char c ;`	`printf("%c",c) ;`	Caractère
`float v ;`	`printf("%f",v) ;`	Point décimal
`float v ;`	`printf("%e",v) ;`	Exponentiel (virgule flottante)
`float v ;`	`printf("%E",v) ;`	

6. Syntaxe de scanf () : lecture de variables de type `float` et `int`

Soit à traduire en langage C l'instruction suivante écrite en langage algorithmique :

Lire (**A**) où **A** est de type **Réel** (`float`).

La traduction en C s'écrit : `scanf ("%f", &A) ;`

Précise le format d'acquisition de la variable saisie au clavier. Ce format doit **impérativement** respecter le **type** de la variable indiqué lors de sa déclaration.

ATTENTION : Il faut transmettre **l'adresse** de la variable à lire à l'instruction scanf (ici **A** dont l'adresse s'écrit **&A**). Risque de graves dysfonctionnement si l'opérateur "adresse" **&** est oublié !

Soit à traduire en langage C l'instruction suivante écrite en langage algorithmique :

Lire (**i**) où **i** est de type **Entier** (`int`).

La traduction en C s'écrit : `scanf ("%d", &i) ;`

Précise le format d'acquisition de la variable saisie au clavier. Ce format doit **impérativement** respecter le **type** de la variable indiqué lors de sa déclaration.

✓ Mettre en commentaire les instructions d'affectations écrites en début de programme et effectuer la saisie en langage C de la séquence algorithmique suivante :

Ecrire ("Valeur de A ? ")

Lire (A)

Ecrire ("Valeur de i ? ")

Lire (i)

✓ Ecrire les instructions d'édition en C afin d'obtenir, dans le cas où l'utilisateur saisit au clavier les valeurs 15.25 et 150, l'écran d'exécution suivant :

```
Premier programme

Valeur de A ? 15.25
Valeur de i ? 150

A contient : 15.250000
i contient : 150

Appuyer sur une touche pour continuer ...
```

7. Structure alternative

7.1. Exemple

Dans la partie édition du programme écrit dans le fichier **`main_ecriture_lecture.c`** , ajouter la traduction C des instructions d'édition suivante écrites en langage algorithmique :

Si (A < 0)
  Alors Ecrire ("A < 0 ")
  Sinon Ecrire ("A >= 0 ")
Fin si

Tester le programme.

7.2. Résolution d'une équation du premier degré

✓ Créer un **nouveau** fichier projet **`Prj_eq_deg1.dev`** et enregistrer le fichier **`main.c`** (nom par défaut) sous le nom **`main_eq_deg1.c`**. Vérifier que ce fichier est bien associé au projet.

✓ Supprimer les arguments de la fonction **`main()`**.

On souhaite saisir un programme permettant de résoudre l'équation du premier degré $b\,x + c = 0$ en envisageant tous les cas de figure et en séparant la saisie, le traitement et l'édition des résultats (cf. **TD n°1**).

VAR

B, C : Réel

X : Réel

Test : Entier

```
Début
    /*Saisie des coefficients */
    Ecrire ("Resolution de B x + C = 0")
    Ecrire ("Valeur de B ?")
    Lire ( B )
    Ecrire ("Valeur de C ?")
    Lire ( C )
  /* Traitement */
  Si ( B = 0)
     Alors  Si (C = 0)
                Alors   Test ← 0
                Sinon   Test ← 1
            Fin si
     Sinon  Test  ← 2
            X  ←  – C  / B
  Fin si
  /* Edition */
   Ecrire ("L'équation ", B , "x + " , C , " = 0 ")
   Cas de  (Test)
      0 : Ecrire ("admet une infinité de solutions")
      1 : Ecrire ("n'admet pas de solution")
      2 : Ecrire ("admet une solution : x =", X)
   Fin cas
  Fin
```

✓ Développer complètement le programme d'équation du premier degré.

On souhaite obtenir à l'écran des résultat d'exécution tel que :

```
Resolution de B x + C = 0

Valeur de B ? 30
Valeur de C ? 1

L'equation 30.00 x + 1.00 = 0
admet une solution x = -3.33E-002

Appuyer sur une touche pour continuer . . .
```

```
Resolution de B x + C = 0

Valeur de B ? 0
Valeur de C ? 4

L'equation 0.00 x + 4.00 = 0
n'admet pas de solution

Appuyer sur une touche pour continuer . . .
```

```
Resolution de B x + C = 0

Valeur de B ? 0
Valeur de C ? 0

L'equation 0.00 x + 0.00 = 0
admet une infinite de solutions

Appuyer sur une touche pour continuer . . .
```

Remarque : Il est possible de contrôler le format d'affichage de valeurs numériques réelles, soit en virgule fixe, soit en virgule flottante :

✓ Affichage de la variable **A** (de type **`float`**) en virgule fixe, si celle-ci contient la valeur **`0.015390`**

Instructions	Sortie écran correspondante

`printf("A = %f",A) ;`
`printf("A = %.2f",A) ;`
`printf("A = %.4f",A) ;`
	`A = 0.015390`
	`A = 0.02`
	`A = 0.0154`

✓ Affichage de la variable **A** (de type **`float`**) en virgule flottante, si celle-ci contient la valeur **`0.0153`**

Instructions	Sortie écran correspondante

`printf("A = %E",A) ;`
`printf("A = %.1E",A) ;`
`printf("A = %.2E",A) ;`
	`A = 1.539000E-002`
	`A = 1.5E-002`
	`A = 1.54E-002`

Il est en plus possible de réserver le nombre de caractères dédiés à l'affichage du contenu des valeurs. La première valeur numérique située après **%** indique le nombre total de caractères réservés pour afficher le résultat. (*Si ce nombre ne suffit pas, printf rajoute automatiquement le nombre de caractères nécessaire*s).
Exemples :

Instructions	Sortie écran correspondante
`// si i (int) contient 1539`	

`printf("i = %d",i)  ;`
`printf("i = %6d",i) ;`
`printf("i = %10d",i);`
	`i = 1539`
	`i =   1539`
	`i =       1539`

Instructions	Sortie écran correspondante

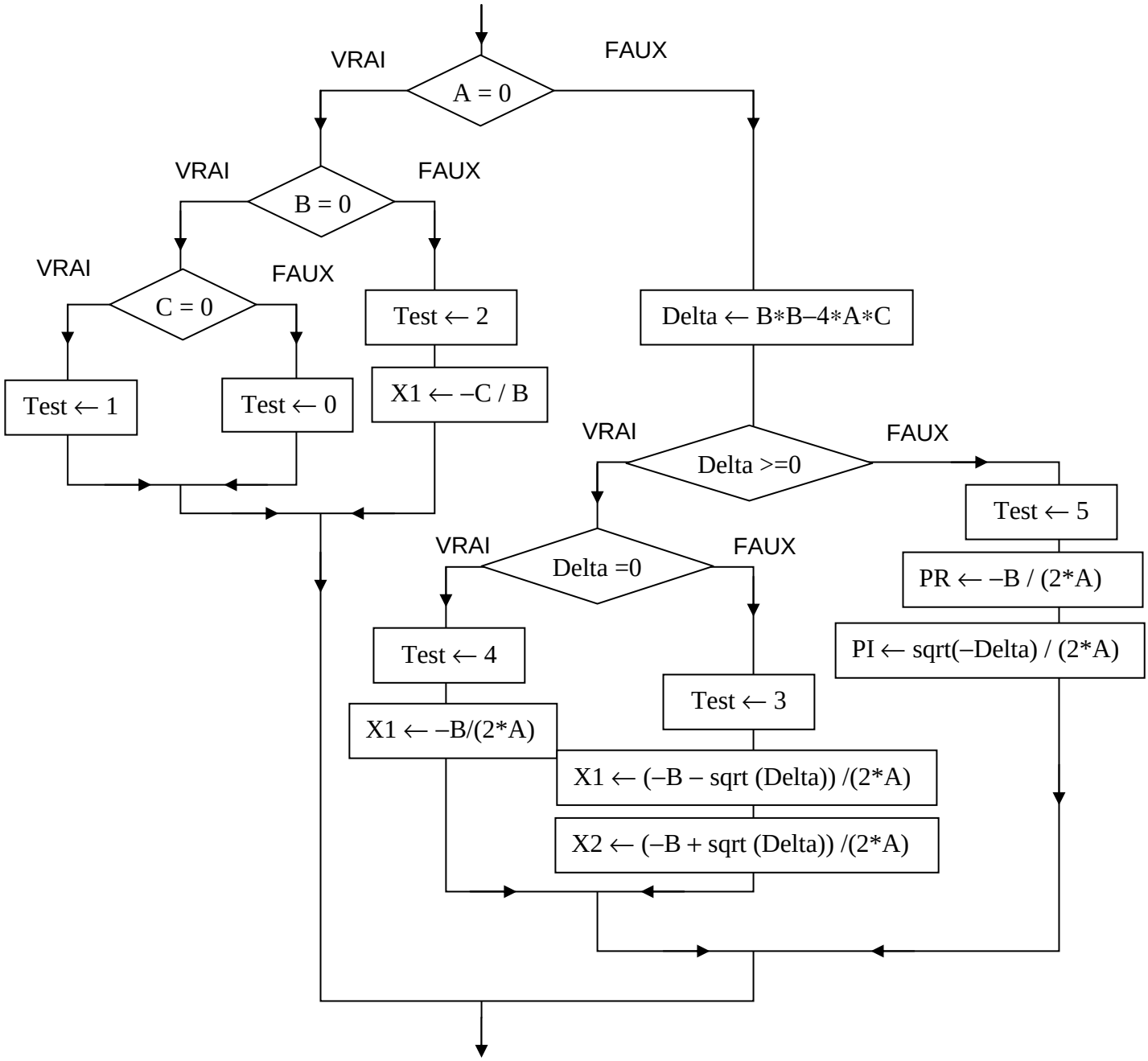
`printf("A = %f",A)     ;`
`printf("A = %5.2f",A)  ;`
`printf("A = %7.2f",A)  ;`
`printf("A = %E",A)     ;`
`printf("A = %10.2E",A) ;`
`printf("A = %14.2E",A) ;`
	`A = 0.015390`
	`A =  0.02`
	`A =    0.02`
	`A = 1.539000E-002`
	`A =  1.54E-002`
	`A =      1.54E-002`

Utiliser les options d'affichage adaptées pour obtenir les écrans d'exécution souhaités.

TRAVAUX SUPPLEMENTAIRES

8. Resolution d'une équation du second degré A x² + B x + C = 0

L'organigramme de **traitement** possible est fourni ci-dessous :



✓ Rédiger le programme en langage algorithmique (**page 14**) en prévoyant **3 parties** de programme séparées :

- la saisie des coefficients,
- le traitement,
- l'édition des résultats.

✓ Créer un nouveau projet que vous appellerez **`Prj_eq_deg2.dev`** dans lequel vous développerez un fichier source **`main_eq_deg2.c`** permettant de résoudre l'équation du second degré : $Ax^2 + Bx + C = 0$.

✓ Ecrire alors la traduction en langage C de ce programme dans le fichier **`main_eq_deg2.c`**. (inclure le header **`<math.h>`** pour utiliser l'instruction **`sqrt()`**).

On souhaite obtenir à l'écran des résultats d'exécution tels que :

```
Resolution de A x^2 + B x + C = 0

Valeur de A ? 1
Valeur de B ? -3
Valeur de C ? 2

L'equation (1.00) x^2 +(-3.00) x + (2.00) = 0
admet deux racines reelles distinctes :
   X1 = 1.00E+000
   X2 = 2.00E+000

Appuyer sur une touche pour continuer . . .
```

```
Resolution de A x^2 + B x + C = 0

Valeur de A ? 1
Valeur de B ? -2
Valeur de C ? 1

L'equation (1.00) x^2 +(-2.00) x + (1.00) = 0
admet une racine double :
   X1 = X2 = 1.00E+000

Appuyer sur une touche pour continuer . . .
```

```
Resolution de A x^2 + B x + C = 0

Valeur de A ? 1
Valeur de B ? 1
Valeur de C ? 1

L'equation (1.00) x^2 + (1.00) x + (1.00) = 0
admet deux racines complexes conjuguées :
   X1 = -5.00E-001 + j (8.66E-001)
   X2 = -5.00E-001 - j (8.66E-001)

Appuyer sur une touche pour continuer . . .
```

```
Resolution de A x^2 + B x + C = 0

Valeur de A ? 0
Valeur de B ? 1
Valeur de C ? 1

L'equation (0.00) x^2 + (1.00) x + (1.00) = 0
admet une racine simple reelle :
   X1 = -1.00E+000

Appuyer sur une touche pour continuer . . .
```

```
Resolution de A x^2 + B x + C = 0

Valeur de A ? 0
Valeur de B ? 0
Valeur de C ? 1

L'equation (0.00) x^2 + (0.00) x + (1.00) = 0
n'admet pas de solution

Appuyer sur une touche pour continuer . . .
```

```
Resolution de A x^2 + B x + C = 0

Valeur de A ? 0
Valeur de B ? 0
Valeur de C ? 0

L'equation (0.00) x^2 + (0.00) x + (0.00) = 0
a une infinite de solutions

Appuyer sur une touche pour continuer . . .
```

Rédaction du programme en langage algorithmique

VAR

Début

/* Traitement */

/* Edition des résultats */

9. Validation d'une date

Ecrire un programme permettant de tester la validité d'une date saisie au clavier. La date sera saisie à l'aide de 3 variables **Jour**, **Mois**, **Annee** de type **Entier** .

On utilisera une variable **Valid** qui vaudra **1** si la date est valide et **0** dans le cas contraire .

Il faudra contrôler que :

- **1** <= **Mois** <= **12**
- **1** <= **Jour** <= **31** pour les mois à 31 jours
- **1** <= **Jour** <= **30** pour les mois à 30 jours

✓ Cas du mois de Février : (**Mois** = **2**)

Une année est dite **bissextile** (du latin bissextilis) lorsqu'elle comporte **366** jours au lieu de **365**. Cette fameuse année revient tous les **4** ans et comprend un **29** février.

Règles :

- Les années **divisibles par 4** sont **bissextiles**, pas les autres.
- Exception: les années **divisibles par 100** ne sont pas bissextiles.
- Exception à l'exception: les années **divisibles par 400** sont bissextiles.

Rédaction du programme en langage algorithmique

VAR

Début

/* Saisie de la date */

/* Traitement */

.....

.....

.....

.....

.....

/* Edition */

.....

.....

.....

.....

.....

.....

.....

Fin

✓ Créer un projet **prj_date.dev** et traduire l'algorithme en langage C dans un fichier **main_date.c**.

ITERATIONS

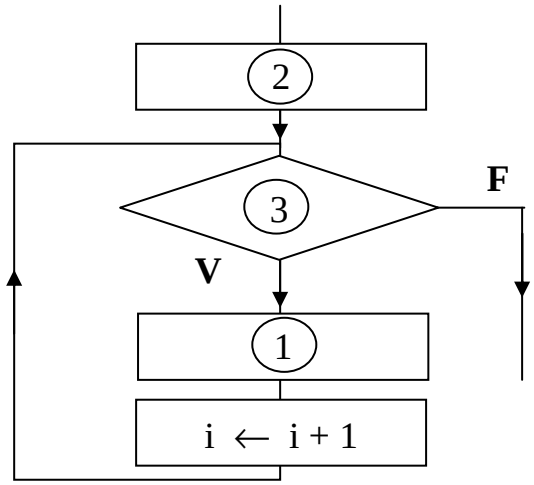
Thème abordé : *structures itératives (manuel* **Eléments de base du langage C**, *p.14-15)*

1. Méthodologie d'analyse d'une structure itérative

1 Définir **l'action qui devra être répétée**.
Dans le cas d'un calcul récursif, écrire le calcul (instruction d'affectation) a l'itération n° **i** en fonction de l'itération précédente.

2 Déduire, à partir de de l'instruction écrite à l'itération n° **i**, les instructions d'initialisation (de la boucle et du calcul).

3 Définir la condition logique pour rester dans la boucle (afin de répéter le traitement).



2. Ecriture en langage C de diverses formes d'itérations

2.1. Exercice n°1 : somme des N premiers entiers

Ecrire les instructions de calcul de la somme des **N** premiers entiers que l'on stokera dans la variable **Somme**. On utilisera la variable **i** pour l'indice de boucle.

Somme = 1 + 2 + ….. + (i-1) + i + (i+1) + ….. + N

1°) Ecrire l'instruction qui devra être répétée : ……..…….

2°) Ecrire les instructions d'initialisation : ……..……..……

……..…….

3°) Ecrire la condition logique de sortie de boucle : ……...

Déduire de cette analyse la séquence en langage alogoritmique utilisant l'itération universelle **Tant que** permettant d'effectuer le calcul demandé :

……..

……..

……..

……..

……..

……..

2.2. Exercice n°2 : somme des N premiers entiers carrés

Ecrire les instructions de calcul de la somme des **N** premiers entiers **carrés** que l'on stokera dans la variable **Somme_carre**. On utilisera la variable **i** pour l'indice de boucle.

Somme_carre = 1*1 + 2*2 + ….. + (i-1)* (i-1)*+ i*i + (i+1)*(i+1) + ….. + N*N

1°) Ecrire l'instruction qui devra être répétée : ……..…….

2°) Ecrire les instructions d'initialisation : ……..……..……

……..…….

3°) Ecrire la condition logique de sortie de boucle : ……...

Déduire de cette analyse la séquence en langage alogoritmique utilisant une boucle **Pour** permettant d'effectuer le calcul demandé :

……..

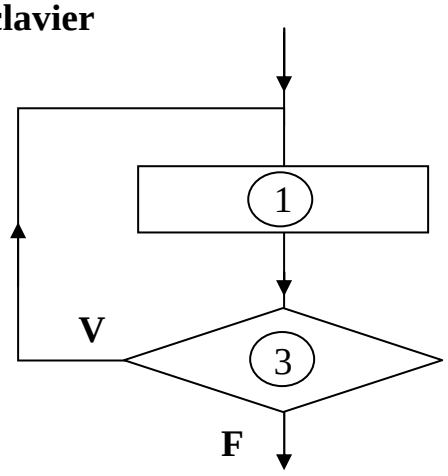
……..

……..

……..

2.3. Exercice n°3 : détection d'une touche clavier

Ecrire une boucle d'attente qui effectue la saisie d'une touche au clavier tant que l'utilisateur n'a pas saisi le caractère **`'c'`**.



Remarque : la structure itérative envisagée est représentée par l'organigramme ci-contre. Contrairement aux 2 structures précédentes vues aux paragraphes 2.1 et 2.2, la phase d'initialisation n'est pas nécessaire.

Pour effectuer la lecture d'une variable de type **caractère**, on utilisera en langage C l'instruction :

```
Touche_pressee = getche() ;
```

……..

……..

……..

……..

✓ Charger le fichier projet **`Prj_iter.dev`** situé dans votre répertoire **`U:\Prog_c\TP2`**.
Editer le fichier **`main_iter.c`** dont le listing est fourni ci-dessous :

```
/*
   Nom de fichier disque : main_iter.c
   Projet : Prj_iter.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : Utilisation des structures itératives
                 Calcul de la somme des N premiers entiers
   Date :
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main()
{
   // Déclaration des variables du programme principal

   int  i, N
   int  Somme, Somme_carre ;
   char Touche_pressee     ;

   // Corps exécutable du programme principal

   printf("Entrer la valeurs de N : ") ;
   scanf("%d",&N) ;

   // Exercice n°1 : Somme des N premiers entiers
   // (structure universelle Tant que () Faire ..... Fin tant que

   // Exercice n°2 : Somme des N premiers entiers carrés
   // (structure boucle Pour)

   // Exercice n°3 : Lecture d'une touche au clavier
   // (Utiliser la fonction getche() )

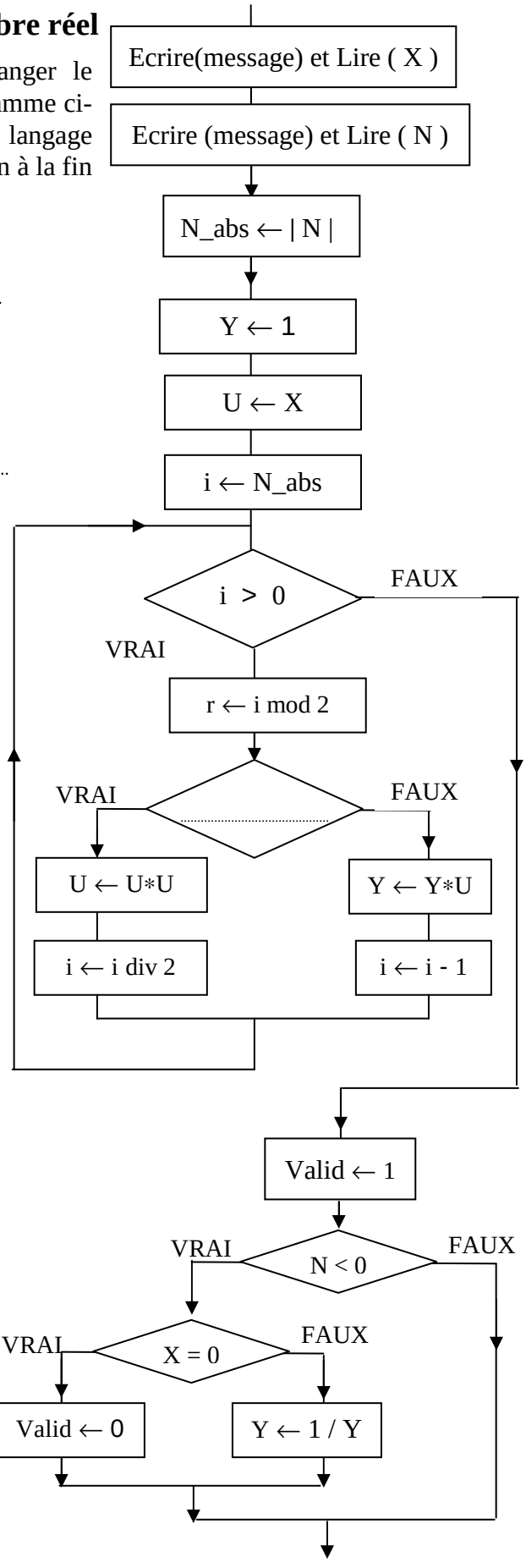
   // Edition des résultats

   printf("\n") ;
   system("PAUSE");
   return 0;
}
```

Exécuter le programme en mode pas à pas et contrôler le contenu des variables (voir Annexe p.4)

3. Calcul de la puissance entière d'un nombre réel

✓ l'algorithme permettant de calculer $\mathbf{X^N}$ et de ranger le résultat dans la variable **Y** est fourni sur l'organigramme ci-contre. Traduire cet organigramme en langage algorithmique en prévoyant les instructions d'édition à la fin du bloc exécutable.



VAR

.....

.....

.....

.....

Début

..... /* Saisie */

.....

.....

.....

..... /* Traitement */

/* Calcul de la valeur absolue */.....

Si .....

Alors .....

Sinon .....

Fin si .....

/* Initialisations */.....

.....

.....

.....

.....

.....

.....

/* Calcul de X^N_abs */

/* Calcul de X^N */

/* Edition des résultats */

✓ Créer un nouveau projet **`Prj_puis.dev`** contenant un fichier **`main_puis.c`**. Traduire en C l'algorithme précédent et le tester en mode pas à pas.

On souhaite obtenir à l'écran des résultats d'exécution tels que :

```
Calcul de Y = X^N

Valeur de X ? 2
Valeur de N ? -2

2.00^(-2) = 2.50E-001
Appuyer sur une touche pour continuer . . .
```

```
Calcul de Y = X^N

Valeur de X ? 0
Valeur de N ? -2

0.00^(-2) n'existe pas
Appuyer sur une touche pour continuer . . .
```

```
Calcul de Y = X^N

Valeur de X ? 5
Valeur de N ? 0

5.00^(0) =  1.00E+000
Appuyer sur une touche pour continuer . . .
```

TRAVAUX SUPPLEMENTAIRES

4. Calcul du pgcd de deux nombres entiers

Le pgcd de deux nombres entiers strictement positifs A et B est le **p**lus **g**rand **c**ommun **d**iviseur que l'on note pgcd(A, B).

Exemple : pgcd(70,45) = 5

Propriétés : pgcd (A, A) = A, pgcd (A, 0) = A.

Il est ici proposé d'écrire deux programmes correspondant à deux algoritmes de calcul.

4.1. Technique des soustractions successives

Principe : Il repose sur la propriété arithmétique suivante :

> Si A < B, alors pgcd (A, B) = pgcd (A, B–A)
>
> Il est permis de remplacer le plus grand des deux nombres par la différence des deux nombres; ceci ne change pas le pgcd.

Il suffit donc de considérer les paires d'entiers successives en gardant le plus petit et en soustrayant le plus petit du plus grand. Le pgcd est obtenu lorsque les deux entiers sont égaux.

Exemple :

pgcd(70,45) = pgcd (70-45,45) = pgcd (25,45)

pgcd (25,45) = pgcd (25,45-25) = pgcd (25,20)

pgcd (25,20) = pgcd (25-20,20) = pgcd (5,20)

pgcd (5,20) = pgcd (5,20-5) = pgcd (5,15)

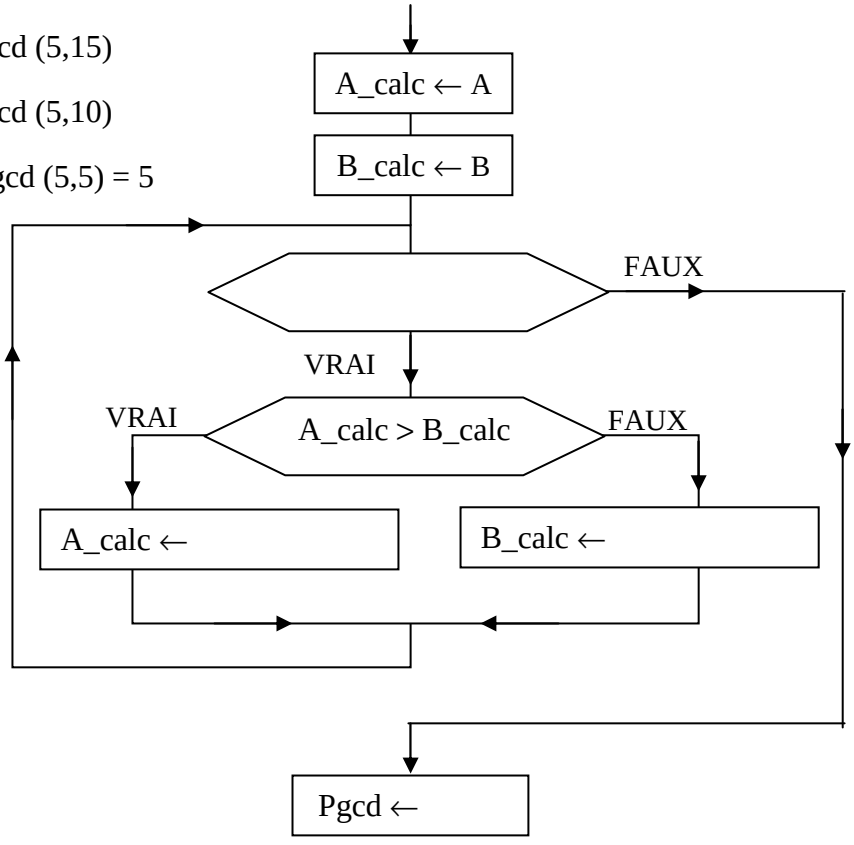
pgcd (5,15) = pgcd (5,15-5) = pgcd (5,10)

pgcd (5,10) = pgcd (5,10-5) = pgcd (5,5) = 5

Le calcul se termine lorsque les valeurs dont on cherche itérativement le pgcd deviennent égales.

L'algorithme de traitement ci-contre utilise deux variables auxiliaires **A_calc** et **B_calc** .

✓ Compléter cet organigramme et le traduire en langage algorithmique.



Développement du programme en langage algorithmique :

VAR

.....

.....

.....

.....

Début

/* Saisie des opérandes A et B */

.....

.....

.....

.....

/* Traitement */

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

/* Edition des résultats */

.....

.....

Fin

✓ Créer un nouveau projet **`Prj_pgcd.dev`** contenant un fichier **`main_pgcd_soustr.c`**.

Celui-ci prévoira la saisie des valeurs **a** et **b**, le traitement et l'édition des résultats.

On souhaite obtenir à l'écran des résultats d'exécution tels que :

```
Calcul du pgcd de deux entiers positifs A et B

Valeur de A ? 525
Valeur de B ? 375

pgcd(525,375) = 75

Appuyer sur une touche pour continuer . . .
```

4.2. Algorithme d'Euclide : technique des divisions successives

Principe : Il repose sur la propriété arithmétique suivante :

> Si A > B, alors pgcd (A, B) = pgcd (B, R) si R est le reste de la division entière de A par B.
>
> Il est permis de remplacer le plus grand des deux nombres par le reste de la division euclidienne du plus grand nombre par le plus petit ; ceci ne change pas le pgcd.

Il suffit donc de considérer les paires d'entiers successives en gardant le plus petit et en prenant le reste de la division entière du plus grand par le plus petit du plus grand. Le pgcd est obtenu lorsque le reste devient nul.

Exemple :

70 : 45 = 1 reste 25
pgcd(70,45) = pgcd (25,45)

45 : 25 = 1 reste 20
pgcd (25,45) = pgcd (25,20)

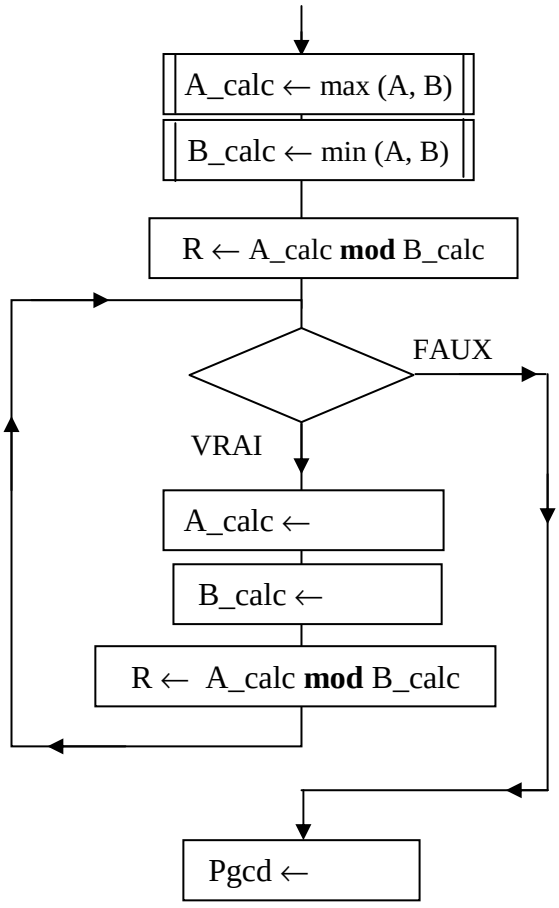
25 : 20 = 1 reste 5
pgcd (25,20) = pgcd (5,20)

20 : 5 = 4 reste 0
pgcd (5,20) = pgcd (5,0) = 5

Cet algorithme est plus rapide que celui utilisant la technique des soustractions successives.

L'algorithme d'Euclide utilisant deux variables auxiliaires **A_calc** et **B_calc** est déterminé par l'organigramme ci-contre.

✓ Compléter cet organigramme et le traduire en langage algorithmique.



Développement du programme en langage algorithmique :

VAR

.....

.....

.....

.....

Début

/* Saisie des opérandes A et B */

.....

.....

.....

.....

/* Traitement */

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

/* Edition des résultats */

.....

Fin

✓ Créer un nouveau un nouveau projet **`Prj_pgcd_euclide.dev`** contenant un fichier fichier **`main_pgcd_euclid.c`** .

Celui-ci prévoira la saisie des valeurs **A** et **B**, le traitement et l'édition des résultats.

Le résultat à l'écran est identique au cas précédent.

5. Développement limité de la fonction exp(*x*)

La fonction mathématique **exp (*x*)** peut s'approcher par un polynôme $\mathbf{P_N}(x)$ d'ordre **N** :

$$\mathbf{exp}(x) \cong \mathbf{P_N}(x) = 1 + x + \ldots + \frac{x^{\mathbf{i}}}{\mathbf{i}!} + \ldots \frac{x^{\mathbf{N}}}{\mathbf{N}!} \qquad \textbf{(1)}$$

Le polynôme $\mathbf{P_N}(x)$ s'écrit donc sous la forme :

$$\mathbf{P_N}(x) = \sum_{\mathbf{i}=0}^{\mathbf{N}} \frac{x^{\mathbf{i}}}{\mathbf{i}!} = \sum_{\mathbf{i}=0}^{\mathbf{N}} \mathbf{Term_i}(x) \qquad \textbf{(2)}$$

avec $\mathbf{Term_i}(x) = \frac{x^{\mathbf{i}}}{\mathbf{i}!}$ **(3)**

La **précision obtenue** du calcul, à l'ordre **N**, se caractérise par la valeur absolue de la **variation relative** :

$$\varepsilon_{\mathbf{N}} = \left| \frac{\mathbf{P_N}(x) - \mathbf{P_{N-1}}(x)}{\mathbf{P_{N-1}}(x)} \right| \qquad \textbf{(4)}$$

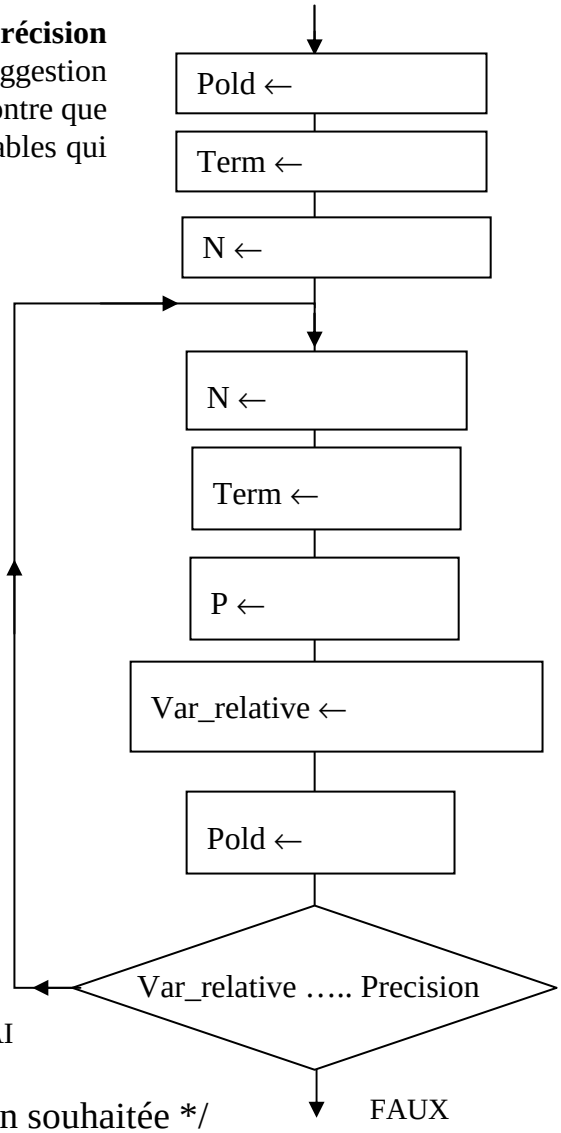
On obtient donc un calcul approché de **exp (*x*)** par un polynôme d'ordre **N** avec une précision souhaitée η *si* $\varepsilon_N < \eta$ ➔ **variation relative < précision souhaitée**.

1°) Montrer que $\mathbf{Term_i}(x) = \mathbf{K}\ \mathbf{Term_{i-1}}(x)$. Donner l'expression de **K** en fonction de ***x*** et **i**.

2°) Après avoir effectué la saisie de ***x*** et de la **précision du calcul souhaitée** (en **%**), dans la variable Precision, écrire l'algorithme permettant :

- le calcul de **exp(*x*)** à l'aide d'un polynôme,
- l'ordre **N** (> 1) du polynôme pour obtenir la **précision souhaitée**.

On écrira les instructions d'édition de **N**, de la **précision obtenue** (en %) ainsi que la valeur $\mathbf{P_N(x)}$. Une suggestion pour le traitement est fourni sur l'organigramme ci-contre que l'on complètera en diquant le rôle et le type des variables qui y figurent.



Développement du programme en langage algorithmique :

VAR

.....

.....

.....

.....

.....

.....

.....

.....

Début

/ * Saisies de la valeur x et de la précision souhaitée */

.....

.....

.....

.....

/* Traitement */

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

/* Edition des résultats */

.....

.....

.....

.....

Fin

3°) Développer ce programme en C dans le fichier **`main_exp.c`** en créant un projet **`Prj_DL_exp.dev`**.

On comparera le résultat $\mathbf{P_N(x)}$ obtenu avec le calcul obtenu à l'aide de la fonction **exp(x)** de la bibliothèque **<math.h>** en éditant les valeurs en format virgule flottante standard.

Un exemple d'écran d'exécution est fourni ci-dessous :

```
Entrer la valeur de x : 1
Entrer la precision desiree : (en pourcent) 0.1

exp ( 1.00) peut etre approche par un polynome d'ordre 6
avec une precision relative +/- (5.11E-002) %


exp ( 1.00) = 2.7182818E+000


P( 1.00) = 2.7180557E+000

Appuyez sur une touche pour continuer... _
```

Pour afficher le **%**, utiliser le code 37 ou faire **%%** dans la chaîne de caractères.

TABLEAUX DE NOMBRES

Thème abordé : *types tableaux à une dimension (manuel* **Eléments de base du langage C**, *p.3) , initialisation de variables*

1. Manipulations élémentaires, édition

✓ Charger le projet **`Prj_tableaux.dev`** situé dans votre répertoire **`U:\Prog_c\TP3`**.
Editer le fichier **`main_tab.c`** dont le listing est fourni ci-dessous :

```
/*
   Nom de fichier disque : main_tab.c
   Projet : Prj_tableaux.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : Manipulation des types tableaux
   Date :
*/
#include <stdio.h>
#include <stdlib.h>

#define N_MAX 10      // Définition d'une constante

int main()
{
   // Variables du programme principal

   float Liste1[N_MAX] = {-4.5,-15.5,3.5,5,16.5}; // Variables
   float Liste2[N_MAX] ;                          // tableaux de réels
   int N1 = 5 ;         /* Nombre de valeurs significatives
                           du tableau Liste1
   int N2     ;         /* Nombre de valeurs significatives
                           du tableau Liste2
   int i,k    ;         /* Indices d'accés aux composantes des tableaux
                           Liste1 ou Liste2 */

   // Corps exécutable du programme principal

   /* Instructions permettant de copier dans le tableau Liste2, les
      valeurs du tableau Liste1 qui sont strictement positives */

   // Instruction d'édition des valeurs de Liste1

   // Instruction d'édition des valeurs de Liste2
   printf("\n")   ;
   system("PAUSE");
   return 0       ;
}
```

✓ Quel est le nombre maximal de valeurs pouvant être stockées dans les variables **Liste1** et **Liste2** ?

.....

✓ Quelles sont les valeurs significatives du tableau **Liste1** (consulter les initialisations qui ont été opérées dans la zone déclarative) ?

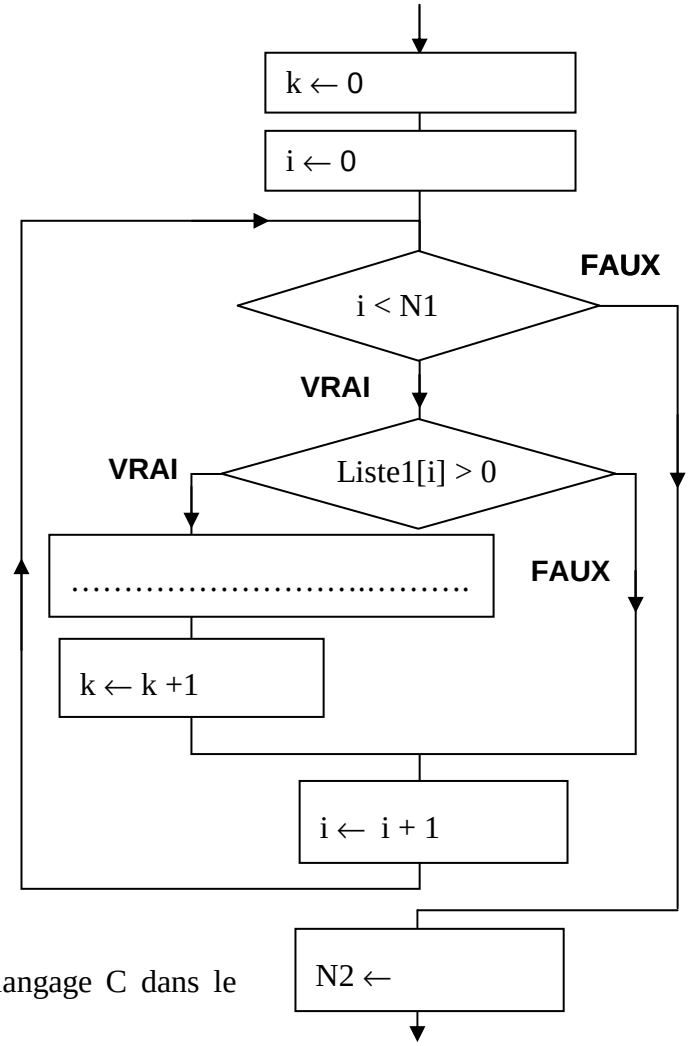
.....

.....

✓ Le tableau **Liste2** contient-il des valeurs significatives ?

.....

✓ A l'aide d'une structure **Pour**, écrire la séquence de traitement en langage algorithmique permettant de copier dans le tableau **Liste2**, les valeurs du tableau **Liste1** qui sont strictement positives. On calculera également le nombre **N2** de valeurs significatives du tableau **Liste2**.



.....

.....

.....

.....

.....

.....

.....

.....

.....

✓ Ecrire les instructions correspondantes en langage C dans le fichier **`main_tab.c`**.

✓ Tester le bon fonctionnement du programme en surveillant le contenu des variables **`N1`**, **`N2`**, **`i`**, **`Liste1[i]`**, **`k`**, **`Liste2[k]`** dans la fenêtre **`Debug`**.

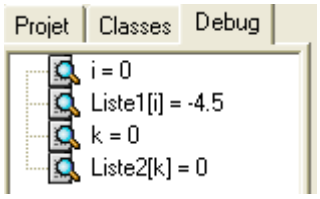
✓ Que peut-on dire de **l'effet de l'initialisation** de la variable **`Liste1`** sur les **5 dernières valeurs** ? Que peut on dire des dernières valeurs de la variable **`Liste2`** ?

.....

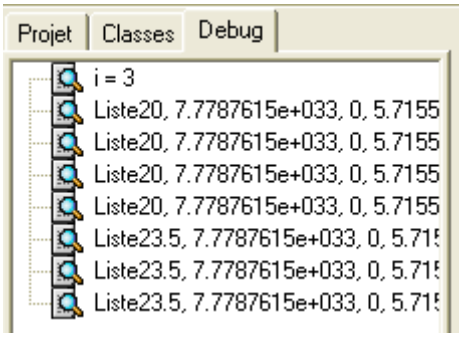
.....

Précautions de débogage de variables tableau.

- Ne tenter de consulter **`Liste1[i]`** ou **`Liste2[k]`** que lorsque **i** et **k** sont affectées de valeurs compatibles avec la taille des tableaux **`Liste1`** ou **`Liste2`**.

Exemple de débogage correct : **$0 \le i \le 9$**, **$0 \le k \le 9$** Le débogueur permet de fournir les contenus **`Liste1[i]`** ou **`Liste2[k]`**	

- On n'a pas intérêt à contrôler une variable tableau **dans sa totalité** en cours de débogage, car cela rajoute un nouveau contenu avec les nouvelles valeurs à chaque fois qu'une nouvelle ligne d'instruction est effectuée. La fenêtre **`Debug`** est alors vite saturée ! (comme indiqué sur la figure ci-contre).



✓ A l'aide du déclarateur **`typedef`**, définir un nouveau type que l'on appellera **`t_reels`** désignant un type tableau d'au plus **`N_MAX`** réels (*voir Manuel "***Eléments de base du langage C***" p.15*) afin de déclarer de manière différente les variables **`Liste1`** et **`Liste2`**. Mettre alors **en commentaires** l'ancienne déclaration.

✓ Ecrire les instructions d'édition des valeurs de chacune des valeurs des tableaux **`Liste1`** et **`Liste2`**. Le résultat d'exécution doit être :

```
Edition des valeurs de Liste1 :

 1eme valeur: -4.50, 2eme valeur:-15.50, 3eme valeur:  3.50, 4eme valeur:  5.00,
 5eme valeur: 16.50,

Edition des valeurs de Liste2 :

 1eme valeur:  3.50, 2eme valeur:  5.00, 3eme valeur: 16.50,

Appuyer sur une touche pour continuer . . .
```

20 caractères

80 colonnes

L'écran d'exécution possède une capacité d'affichage de **80** caractères par ligne. L'affichage d'un élément du tableau utilise donc une chaîne de **20 caractères** dans laquelle se trouvent 2 formats d'affichage. Il faut donc imposer précisément les formats d'affichage afin d'obtenir le résultat souhaité.

2. Opérations statistiques sur les valeurs d'un tableau de nombres

✓ Après avoir rappelé **l'algorithme** de calcul de la valeur moyenne des **`N`** valeurs significatives d'une variable tableau **Liste** d'au plus **N_MAX = 10** réels (vu au TD n°3), écrire les instructions correspondantes en langage C dans le fichier **`main_stat.c`** du projet **`Prj_stat.dev`** dans le répertoire **`U:\Prog_c\TP3`**.

CONST

.....

TYPE

.....

VAR

.....

.....

.....

.....

.....

Début

/* On suppose que les **N = 5** premières valeurs du tableau Liste ont été initialisées */

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Fin

✓ Tester le programme en mode pas à pas à l'aide du débogueur en prévoyant d'initialiser **dès la déclaration** les **N** = **5** premières valeurs de **Liste** tel que **Liste [0] = 8.5**, **Liste [1] = 9.0**, **Liste [2] = 15.5**, **Liste [3] = 18.5**, **Liste [4] = 14.0**.

✓ Ajouter les instructions d'édition pour obtenir un résultat d'exécution de la forme :

```
Edition des valeurs de Liste :

 1ere valeur:  8.50, 2eme valeur: 9.00, 3eme valeur: 15.50, 4eme valeur: 18.50,
 5eme valeur: 14.00,

Valeur moyenne : 13.10

Appuyer sur une touche pour continuer . . .
```

✓ Avant le traitement développé précédemment, rajouter des instructions permettant d'entrer au clavierla valeur de **N** (nombre de valeurs effectives à stocker dans **Liste**) par une saisie **sécurisée**, puis les valeurs de **Liste**.

Le résultat d'exécution pourra être de la forme :

```
Nombre de valeurs a traiter ( <= 10) ? 18
Erreur, recommencer la saisie :  9

1ere valeur : 8.5
2eme valeur : 15.5
3eme valeur : 18
4eme valeur : 16.5
5eme valeur : 13.5
6eme valeur : 14
7eme valeur : 15
8eme valeur : 13
9eme valeur : 7.5

Edition des valeurs de Liste :

 1ere valeur:  8.50, 2eme valeur: 15.50, 3eme valeur: 18.00, 4eme valeur: 16.50,
 5eme valeur: 13.50, 6eme valeur: 14.00, 7eme valeur: 15.00, 8eme valeur: 13.00,
 9eme valeur:  7.50,

Valeur moyenne : 13.50

Appuyer sur une touche pour continuer . . .
```

TRAVAUX SUPPLEMENTAIRES

3. Extraction de la valeur minimale et maximale

✓ Après avoir rappelé l'algorithme du traitement permettant de déterminer les valeurs **minimale** et **maximale** parmi les des **`N`** valeurs significatives de la variable **`Liste`** (vu en TD n°3), écrire les instructions correspondantes en langage C dans le fichier **`main_stat.c`** et les tester en mode pas à pas à l'aide du débogueur. Prévoir les variables supplémentaires nécessaires.

Algorithme de calcul des valeurs minimales et maximales de **Liste**

VAR

.....

.....

.....

.....

.....

Début

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Fin

✓ Ajouter les instructions d'édition nécessaires.

Le résultat d'exécution (*en utilisant les valeurs initiales du tableau* **`Liste`**) pourra être de la forme :

```
Edition des valeurs de Liste :

 1ere valeur:  8.50, 2eme valeur:  9.00, 3eme valeur: 15.50, 4eme valeur: 18.50,
 5eme valeur: 14.00,

Valeur moyenne  : 13.10
Valeur minimale :  8.50
Valeur maximale : 18.50

Appuyer sur une touche pour continuer . . .
```

4. Triangle de Pascal - Tableau à 2 dimensions

La "formule du binôme de Newton" s'exprime par :

$$(a+b)^n = \sum_{p=0}^{n} C_n^p a^{n-p} b^p$$

Les coefficients C_n^p s'appellent les "coefficients binomiaux" ou "coefficients du binôme".

Le triangle de Pascal est le tableau contenant les coefficients du binôme de Newton tel que, à la ligne n et à la colonne p, on inscrit le nombre $C_n^p = \frac{n!}{p!(n-p)!}$ $\quad 0 \le p \le n$

ligne \ *colonne*						
0	**1**					$(a+b)^0 = \mathbf{1}$
1	**1**	**1**				$(a+b)^1 = \mathbf{1}*a + \mathbf{1}*b$
2	**1**	**2**	**1**			$(a+b)^2 = \mathbf{1}*a^2 + \mathbf{2}*a*b + \mathbf{1}*b^2$
3	**1**	**3**	**3**	**1**		$(a+b)^3 = \mathbf{1}*a^3 + \mathbf{3}*a^2*b + \mathbf{3}*a*b^2 + \mathbf{1}*b^3$
4	**1**	**4**	**6**	**4**	**1**	$(a+b)^4 = \mathbf{1}*a^4 + \mathbf{4}*a^3*b + \mathbf{6}*a^2*b^2 + \mathbf{4}*a*b^3 + \mathbf{1}*b^4$

Propriété de récurrence : $n > 0$, $p < n$ $\quad C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$

✓ On considère la déclaration suivante :

CONST

N_MAX = 10

TYPE t_entiers = Tableau [**N_MAX**] [**N_MAX**] d'Entiers

VAR

Binome : t_entiers /* Variable utilisée pour stocker les coefficients du binôme */

n, p : Entier /* Indices d'accès à la variable binome */

En supposant que binome contient les valeurs du triangle de Pascal, compléter :

Binome [0] [0] = ... Binome [4] [2] = ...

Binome [2] [1] = ... Binome [4] [4] = ...

Ecrire l'algorithme permettant :

- de faire la saisie **sécurisée** du nombre désiré **Nlign** (≤ **N_MAX**) de lignes à remplir,
- de calculer les **Nlign** premières lignes du triangle de Pascal. On utilisera la propriété de récurrence des coefficients du binôme.

VAR

.....

.....

.....

.....

Début

/* Saisie sécurisée du nombre de lignes du triangle de Pascal */

.....

.....

.....

.....

.....

.....

.....

/* Calcul du triangle de Pascal */

/* Edition des valeurs du triangle de Pascal */

Fin

✓ Créer un projet **prj_triangle_pascal.dev** et traduire l'algorithme en langage C dans un fichier **main_pascal.c** .

Tester le programme. L'écran d'exécution doit fournir :

```
Nombre de lignes du triangle de Pascal ( <= 10) ? 7

      1
      1     1
      1     2     1
      1     3     3     1
      1     4     6     4     1
      1     5    10    10     5     1
      1     6    15    20    15     6     1

Appuyez sur une touche pour continuer... _
```

✓ Compléter le programme permettant d'afficher à l'écran, à partir du tableau créé précédemment, l'expression du développement de $(\boldsymbol{a} + \boldsymbol{b})^{\mathbf{N}}$. Il faudra prévoir la saisie de la valeur de la puissance **N**, **compatible avec le nombre de lignes Nlign** du triangle de Pascal effectivement calculées au préalable. L'écran d'exécution aura la forme suivante :

```
Nombre de lignes du triangle de Pascal ( <= 10) ? 7

      1
      1     1
      1     2     1
      1     3     3     1
      1     4     6     4     1
      1     5    10    10     5     1
      1     6    15    20    15     6     1

(a+b)^N

valeur de N ( <= 6) ? 4


(a+b)^4 = a^4 + 4*a^3*b^1 + 6*a^2*b^2 + 4*a^1*b^3 + b^4


Appuyez sur une touche pour continuer... _
```

FONCTIONS EN C

Préambule

Afin de distinguer les variables du programme principal (paramètres effectifs) des paramètres formels des sous-programmes (fonctions ou procédures), on adoptera la typographie suivante :

- la première lettre (au moins) de toute variable du programme principal sera en majuscule,
- tous les paramètres effectifs seront en minuscule

1. Introduction

En langage C, une fonction permet de retourner **une valeur** au programme appelant celle-ci. Cette valeur peut alors être directement exploitable dans des instructions d'affectation, d'écriture ou dans des prédicats (tests logiques).

Dans le cadre de ce TP, on étudiera des fonctions n'utilisant que des paramètres formels d'entrée { E }.

On considère la fonction difference, permettant d'effectuer la différence entre deux nombres réels et dont l'algorithme et sa traduction en C sont donnés ci-dessous :

Langage algorithmique	*Langage C*
Appel dans le programme principal :	Appel dans le programme principal :
C ← difference (A, B)	`C = difference(A, B) ;`

Fonction difference (p1 , p2) : Réel **Paramètres formels :** p1, p2 : Réel { E }	`float difference (float p1, float p2)`

Variable locale :
p1_moins_p2 /* variable locale dont le contenu sera retourné */
Début
p1_moins_p2 ← p1– p2
retourner (p1_moins_p2)
Fin
	`{`
	`/* variable locale contenant la valeur à retourner */`
	`float p1_moins_p2 ;`
	`// Corps exécutable de la fonction`
	`p1_moins_p2 = p1- p2 ;`
	`return p1_moins_p2 ;`
	`}`

Remarque : En C, il est nécessaire de déclarer le **prototype** de la fonction avant son utilisation par le programme appelant. Par conséquent, on effectuera cette déclaration en début de programme avant la fonction **`main`** (programme principal).

2. Exemple : fonction difference

On désire développer une fonction permettant de calculer la différence de deux nombres réels.

✓ Ouvrir le projet **`Prj_fonctions.dev`** et compléter le fichier **`main_soustraction.c`** :

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

3°) *Déclaration du prototype de la fonction*

```
int main()
{
   // Variables du programme principal

   float A, B, C ;

   // Corps exécutable du programme principal

   printf("Valeur de A ? ") ;
   scanf("%f",&A) ;
   printf("Valeur de B ? ") ;
   scanf("%f",&B) ;

   C = difference(A, B);

   printf("\nLa difference de %5.2f et de %5.2f vaut : %5.2f\n\n",
             A, B, C) ;
   printf("\n%5.2f -%5.2f = %5.2f\n\n",A,B,difference(A,B)) ;
   printf("\n")    ;
   system("PAUSE") ;
   return 0;
}
```

1°) *Appel de la fonction* `difference`

```
// Définir la fonction différence ici
```

2°) *Définition de la fonction (à compléter)*

Les différentes étapes de la méthode de développement consistent à :

1°) ou 2°) Ecrire l'appel dans le programme principal (fonction principale **`main`**)

2°) ou 1°) Définir la fonction après le programme principal

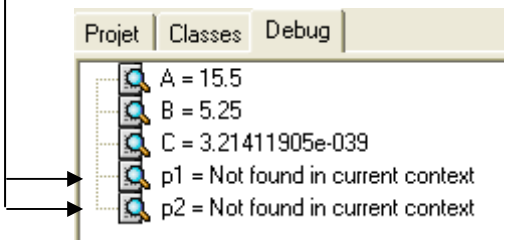
3°) Déclarer le prototype de la fonction **avant** le début de la fonction principale en recopiant l'en-tête de la fonction écrite lors de sa définition (sans oublier le **`;`**)

Important : donner obligatoirement aux paramètres formels des noms en minuscules et différents de ceux des paramètres effectifs !

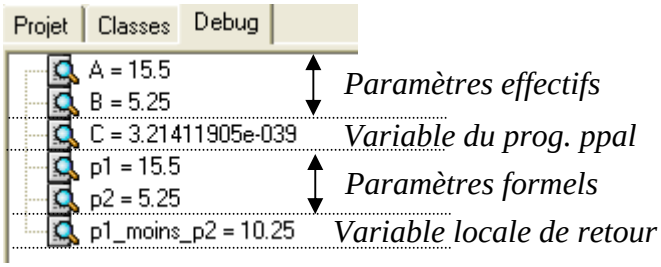
✓ Tester le programme en mode pas à pas [Avancer] (**F7**) afin de suivre le déroulement de la fonction **`difference`** en mode pas à pas.

✓ Contrôler le contenu des variables du programme principal **`A`**, **`B`** et **`C`**.

Remarque : Lorsqu'on exécute le mode pas à pas **dans le programme principal**, le contenu des paramètres formels n'est pas accessible, car la fonction n'a pas démarré son éxécution et le message :
`Not found in current context` apparaît dans la fenêtre **Debug** .

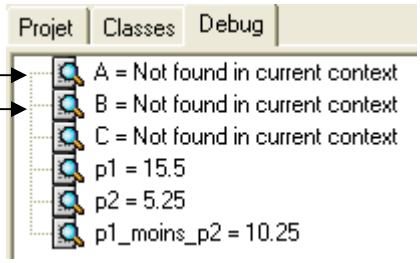


✓ Contrôler le contenu des paramètres formels **`p1`**, **`p2`** au cours du déroulement de l'exécution de la fonction



`A =` .....
`B =` .....
`C =` .....
`p1 =` .....
`p2 =` .....
`p1_moins_p2 =` .....

Remarque : Le contenu des variables du programme principal ne sont plus accessibles à l'intérieur de la fonction et l'on observe le message :
`Not found in current context`



Afin de revoir le contenu des variables **A**, **B**, **C,** il suffit de refaire ([Ajout Variable]) lorsqu'on revient dans le programme principal.

3. Exercices de base de développement de fonctions

On développera les fonctions demandées dans la série d'exercices dans le fichier **`main_exos_fonctions.c`** du projet **`Prj_exos_fonctions.dev`**.

3.1. Fonction min

Compléter en langage algorithmique la fonction **min** permettant d'évaluer la valeur minimale de deux valeurs réelles. Si **V_min** est une variable de type réel, un exemple d'appel dans le programme principal est :

V_min ← min (–25.2 , 105.4)

Fonction min (p1 , p2) : .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

- ✓ Développer la fonction **`min`** dans le fichier **`main_exos_fonctions.c`** et la tester en mode pas à pas.
- ✓ Proposer l'édition du résultat en utilisant la variable **`V_min`** du programme principal, puis sans l'utiliser dans l'instruction d'édition **`printf`**.

Dans les deux cas, le résultat d'exécution devra être conforme à l'affichage ci-dessous :

```
La valeur minimale entre –25.2 et 105.4 vaut -25.2
```

3.2 Fonction max

Compléter en langage algorithmique la fonction max permettant d'évaluer la valeur maximale de deux valeurs réelles. Si **A**, **B** et **V_max** sont 3 variables de type réel (il faudra affecter des valeurs à **A** et **B**), un exemple d'appel est :

V_max ← max (**A**, **B**)

Fonction max (p1 , p2) : .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

- ✓ Développer la fonction **`max`** dans le fichier **`main_exos_fonctions.c`** et la tester en mode pas à pas.
- ✓ Proposer l'édition du résultat en utilisant la variable **`V_max`** du programme principal, puis sans l'utiliser dans l'instruction d'édition **`printf`**.

  Dans les deux cas, le résultat d'exécution devra être conforme à l'affichage ci-dessous (pour **`A = -5.2`** et **`B = 10.5`**) :

```
La valeur maximale entre -5.20 et 10.50 vaut 10.50
```

3.3 Fonction valeur absolue

Compléter en langage algorithmique la fonction **valeur_absolue** permettant de calculer la valeur absolue d'une valeurs réelle. Si **A** et **Abs_A** sont deux variables de type réel, un exemple d'appel est :

Abs_A ← **valeur_absolue** (**A**) /* Calcul de la valeur absolue de A */

Fonction valeur_absolue (p) : .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

- ✓ Développer la fonction **`valeur_absolue`** dans le fichier **`main_exos_fonctions.c`** et la tester en mode pas à pas.
- ✓ Editer le résultat permettant d'afficher la valeur absolue de **A**.

Le résultat d'exécution devra être conforme à l'affichage ci-dessous pour **`A = -5.2`** :

```
La valeur absolue de  -5.20 est :  5.20
```

3.4 Fonction majuscule

Compléter en langage algorithmique la fonction **majuscule** permettant de mettre un caractère quelconque en majuscule uniquement si celui-ci est une minuscule. Si **Lettre** désigne une variable de type caractère, des exemples d'appel sont donnés ci-après :

Lettre ← majuscule ('**a**') (Lettre contient alors '**A**')

Lettre ← majuscule ('**?**') (Lettre contient alors '**?**')

Fonction majuscule (lettre_min) :.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

- ✓ Développer la fonction **`majuscule`** dans le fichier **`main_exos_fonctions.c`** et la tester en mode pas à pas.

  Rappel : on utilisera l'instruction **`getche()`** pour faire l'acquisition d'un caractère au clavier

- ✓ Editer le résultat permettant d'afficher la valeur de la variable **`Lettre`**.

  Le résultat d'exécution devra être conforme à l'affichage ci-dessous :

```
La mise en majuscule de 'a' donne 'A'
La mise en majuscule de '?' donne '?'
La mise en majuscule de 'Z' donne 'Z'
```

3.5 Fonction nb_chiffres

Compléter en langage algorithmique la fonction **nb_chiffres** permettant de compter le nombre de chiffres d'un nombre entier strictement positif. Si **Nchif** et **Nombre** sont deux variables de type entier, un exemple d'appel est :

Nchif ⟵ nb_chiffres (**Nombre**)

Afin de déterminer le corps exécutable de la fonction, on complètera l'organigramme permettant de calculer dans la variable locale **nb_c** le nombre de chiffre de l'entier stocké dans **n**.

Fonction nb_chiffres (n) : .....

.....

.....

.....

.....

.....

.....

.....

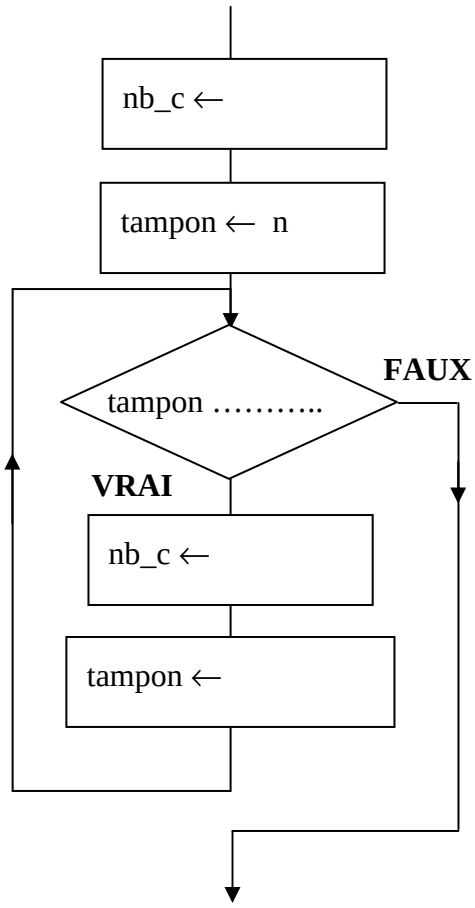
.....

.....

.....

.....

.....



- ✓ Développer la fonction **`nb_chiffres`** dans le fichier **`main_exos_fonctions.c`** et la tester en mode pas à pas.
- ✓ Editer le résultat permettant d'afficher le nombre de chiffres contenu dans un nombre.

  Le résultat d'exécution devra être conforme à l'affichage ci-dessous pour un nombre valant **`145804`** :

```
Le nombre 1458054 contient 7 chiffres.
```

4. Calcul des valeurs de gain et de phase d'un filtre passe-bas

On souhaite écrire un programme permettant de stocker dans des tableaux **Gains** et **Phases** les valeurs du gain (en **dB**) et de la phase (en **degrés**) d'un filtre passe-bas du 1er ordre de fonction de transfert $\mathbf{T(jf) = \frac{A_0}{1 + j\frac{f}{f_c}}}$ pour différentes valeurs de fréquences stockées dans un tableau **Freq** d'au plus **N_MAX** = **18** valeurs réelles.

Le gain est défini par :

$$\mathbf{G(f) = 20 \log |T(jf)|} \text{ avec } \mathbf{|T(jf)| = \frac{|A_0|}{\sqrt{1 + \left(\frac{f}{f_c}\right)^2}}}$$

La phase du filtre s'écrit (en **rad**) :

$\mathbf{\Phi(f) = -\arctan\left(\frac{f}{fc}\right)}$ si $\mathbf{A_0} > 0$

$\mathbf{\Phi(f) = -\arctan\left(\frac{f}{fc}\right) - \pi}$ si $\mathbf{A_0} < 0$

Les fonctions **`log10`** et **`atan`** sont disponibles dans la bibliothèque **`<math.h>`**. Leurs prototypes s'écrivent :

```
float log10 ( float x )
float atan ( float x )
```

Ouvrir le projet **`Prj_passe_bas.dev`** et compléter le fichier **`main_passe_bas.c`** en procédant par étapes définies au §.4.1, 4.2 et 4.3.

4.1. Ecriture de la fonction gain_passe_bas

✓ Compléter en langage algorithmique la fonction **gain_passe_bas** permettant de calculer le gain connaissant les caractéristiques de l'amplification statique $\mathbf{A_0}$, la fréquence de coupure $\mathbf{f_c}$ et la fréquence **f** à laquelle on souhaite connaître le gain.

Si **A0**, **Fc**, **F** et **Gain** sont des variables réelles, l'appel se fera sous la forme :

Gain ← **gain_passe_bas** (**A0**, **Fc**, **F**)

Fonction gain_passe_bas (..... , ... , ...) : .....

.....

.....

✓ Développer la fonction **`gain_passe_bas`** et la tester en mode pas à pas.

4.2. Ecriture de la fonction phase_passe_bas

✓ Compléter en langage algorithmique la fonction **phase_passe_bas** permettant de calculer la phase **en degrés** connaissant les caractéristique de l'amplification statique $\mathbf{A_0}$, la fréquence de coupure $\mathbf{f_c}$ et la fréquence **f** à laquelle on souhaite connaître la phase.

Fonction phase_passe_bas (…….. **,** …….. **,** ……..**) :** ……...

✓ Développer la fonction **`phase_passe_bas`** et la tester en mode pas à pas.

4.3. Ecriture du programme de construction des tableaux Freq, Gains et Phases

✓ Compléter la partie de l'organigramme permettant de calculer les valeurs de fréquence, de gain et de phase dans 3 tableaux **Freq**, **Gains** et **Phases** d'au plus **N_MAX** = **18** à partir des fonctions définies précédemment.

✓ Rédiger l'algorithme correspondant à l'organigramme. Prévoir les instructions d'éditions afin d'obtenir l'écran d'exécution présenté en page suivante.

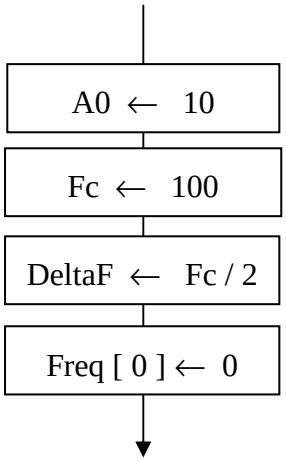
✓ Compléter le programme principal. On effectuera, dans un premier temps, l'initialisation des variables **A0** et **Fc** aux valeurs **10** et **100** pour tester le programme.

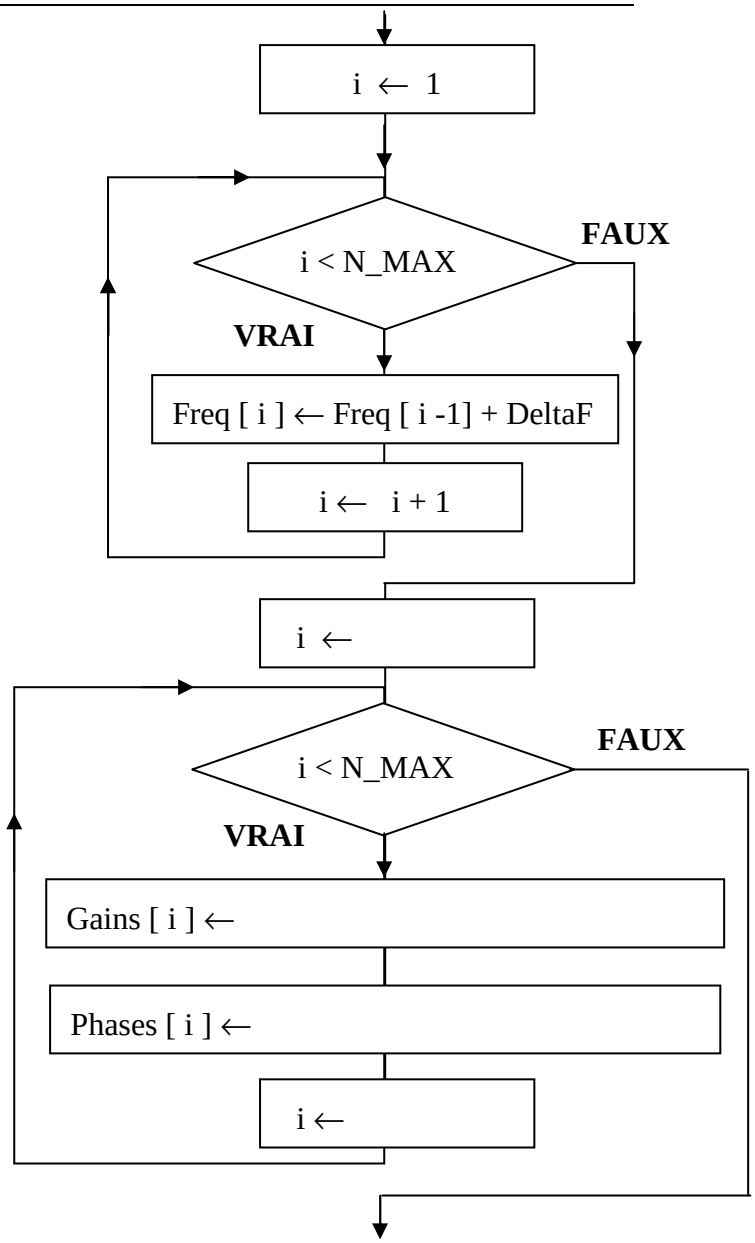
CONST

N_MAX = 18

TYPE t_freq = Tableau [N_MAX] de Réels

VAR





Début

/* Edition des résultats */

Fin

✓ Prévoir les instructions d'éditions afin d'obtenir l'écran d'exécution suivant :

```
Liste des gains en fonction de la frequence :
----------------------------------------------
f =   0.0 Hz ==> G = 20.0 dB             f =   50.0 Hz ==> G = 19.0 dB
f = 100.0 Hz ==> G = 17.0 dB             f =  150.0 Hz ==> G = 14.9 dB
f = 200.0 Hz ==> G = 13.0 dB             f =  250.0 Hz ==> G = 11.4 dB
f = 300.0 Hz ==> G = 10.0 dB             f =  350.0 Hz ==> G =  8.8 dB
f = 400.0 Hz ==> G =  7.7 dB             f =  450.0 Hz ==> G =  6.7 dB
f = 500.0 Hz ==> G =  5.9 dB             f =  550.0 Hz ==> G =  5.1 dB
f = 600.0 Hz ==> G =  4.3 dB             f =  650.0 Hz ==> G =  3.6 dB
f = 700.0 Hz ==> G =  3.9 dB             f =  750.0 Hz ==> G =  2.4 dB
f = 800.0 Hz ==> G =  1.9 dB             f =  850.0 Hz ==> G =  1.4 dB
Liste des phases en fonction de la frequence :
----------------------------------------------
f =   0.0 Hz ==> Phi =   0.0 degres        f =   50.0 Hz ==> Phi = -26.6 degres
f = 100.0 Hz ==> Phi = -45.0 degres        f =  150.0 Hz ==> Phi = -56.3 degres
f = 200.0 Hz ==> Phi = -63.4 degres        f =  250.0 Hz ==> Phi = -68.2 degres
f = 300.0 Hz ==> Phi = -71.6 degres        f =  350.0 Hz ==> Phi = -74.1 degres
f = 400.0 Hz ==> Phi = -76.0 degres        f =  450.0 Hz ==> Phi = -77.5 degres
f = 500.0 Hz ==> Phi = -78.7 degres        f =  550.0 Hz ==> Phi = -79.7 degres
f = 600.0 Hz ==> Phi = -80.5 degres        f =  650.0 Hz ==> Phi = -81.3 degres
f = 700.0 Hz ==> Phi = -81.9 degres        f =  750.0 Hz ==> Phi = -82.4 degres
f = 800.0 Hz ==> Phi = -82.9 degres        f =  850.0 Hz ==> Phi = -83.3 degres

Appuyer sur une touche pour continuer ...
```

TRAVAUX SUPPLEMENTAIRES

5. Calcul du pgcd

✓ Compléter en langage algorithmique la fonction **pgcd** permettant d'évaluer le pgcd de 2 nombres entiers. On utilisera l'algorithme d'Euclide, dont l'organigramme, à compléter, est fourni p.26. Si **PGCD** , **I**, **J** sont 3 variables de type entier, un exemple d'appel est :

PGCD ← pgcd (I, J)

Fonction pgcd (……) :……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

.....

.....

.....

.....

.....

.....

✓ Développer la fonction **pgcd** dans un un nouveau projet et la tester en mode pas à pas.

Editer un résultat dont l'affichage aura la forme :

```
Le pgcd de 1224 et 12280 vaut 8
```

POINTEURS ET PROCEDURES

Thèmes abordés : *Variables pointeurs, passage de paramètres de type simple par valeur ou par adresse (fascicule* **Eléments de base du langage C***, p.19-22)* .

Remarque : Les programmeurs en C ont l'habitude d'appeler par le seul terme **fonction** tous les sous-programmes développés.

Les procédures sont des fonctions **qui ne renvoient aucune valeur** (`void`)

1. Pointeurs

Ouvrir le projet **`Prj_pointeurs.dev`** situé dans votre répertoire **`U:\Prog_C\TP5`** et éditer le fichier **`main_pointeurs.c`** dont le listing est fourni ci-dessous :

```
/*
   Nom de fichier disque : main_pointeurs.c
   Projet : Prj_pointeurs.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : manipulation de variables pointeurs – application
         à la création de paramètres formels de nature {ES} ou {S}
   Date :
*/
#include <stdio.h>
#include <stdlib.h>

int main()
{
   // Variables du programme principal

   float A, B ;
   float aux  ;
   float *x, *y ; // Variables pointeurs

   // Corps exécutable du programme principal

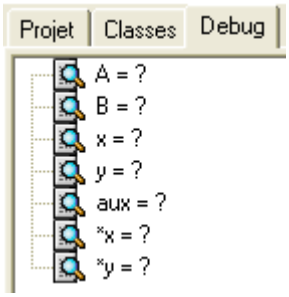
   A = 5.5 ;
   B = 7.5 ;

   /* prise de contrôle des variables A et B
      par les variables x et x */

   x = &A ;
   y = &B ;

   aux = *x ;
   *x = *y  ;
   *y = aux ;
   system("PAUSE") ;
   return 0;
}
```

✓ Tester le programme en mode **pas à pas** à l'aide du **débogueur**. Contrôler en particulier le contenu des variables **A**, **B**, **x**, **y**, **aux** ainsi que le contenu des adresses pointées par **x** (***x**) et **y** (***y**). En déduire **l'adresse** de **A** et celle de **B**.



&A = ...

&B = ...

Que réalisent les instructions effectuées à l'aide des variables **x** et **y** (et de la variable **aux**) ?

.....

.....

.....

Compléter en langage algorithmique la procédure **permute** permettant la permutation du contenu de 2 variables **A** et **B**. L'appel dans le programme principal s'écrit **permute** (**A**, **B**).

2. Ecriture d'une procedure permute

✓ Ecrire en langage algorithmique une procédure **permute** permettant de permuter le contenu de 2 variables **A** et **B** (un exemple d'appel est : **permute** (**A**, **B**)).

Procédure permute (x , y)

.....

.....

.....

.....

.....

.....

.....

.....

.....

✓ Supprimer du projet le fichier **main_pointeurs.c**, puis associer au projet le fichier **main_permute.c** .

```
/*
   Nom de fichier disque : main_permute.c
   Projet : Prj_pointeurs.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : manipulation de variables pointeurs – application
                 à la création de paramètres formels de nature {ES}
   Date :
*/
#include <stdio.h>
#include <stdlib.h>

int main()
{
   // Variables du programme principal

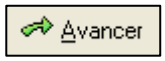
   float A, B ;

   // Corps exécutable du programme principal

   A = 5.5 ;
   B = 7.5 ;

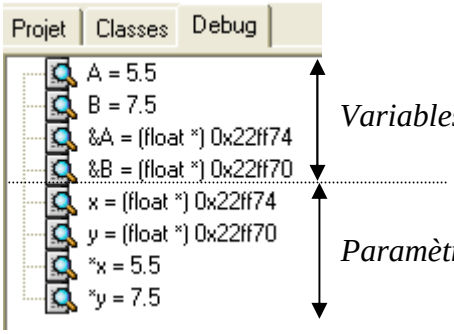
// Ecrire ici l'appel à la procédure permute
// permute (    );
   system("PAUSE") ;
   return 0;
}
```

✓ Développer la procédure **permute** selon la méthode déjà indiquée pour le développement de fonctions à la page 44.

✓ Tester le programme par la commande **F7** 

afin de suivre le déroulement de la procédure **permute** en mode pas à pas.

✓ Contrôler le contenu des variables du programme principal **A**, **B** ainsi que leur adresse (**&A** et **&B**). Contrôler également le contenu des paramètres formels **x**, **y** ainsi que le contenu pointé par ces paramètres (***x**, ***y**).



A = ……

B = ……

&A = ……

&B = ……

x = ……

y = ……

***x =** ……

***y =**……

3. Développement d'un programme de résolution d'une equation du premier degré découpé en modules

On résoudra le problème en écrivant **3 sous-programmes** permettant la saisie des coefficients, le calcul de la racine (si elle existe) et l'édition des résultats.

Le programme principal s'écrit, en langage algorithmique :

```
VAR
    B, C : Réel  /* Coefficient de l'équation */
    X    : Réel  /* Solution quand elle existe */
    Type_soluce : Entier
Début
    saisie_coeff ( B , C )
    resolution_deg1 ( B , C, X , Type_soluce )
    edite_solutions ( B, C , X, Type_soluce )
Fin
```

✓ Compléter en langage algorithmique la procédure **saisie_coeff** permettant de faire la saisie des 2 coefficients **B** et **C** (on précisera le type et la nature des paramètres formels utilisés).

Procédure saisie_coeff (b , c)

.....

.....

.....

.....

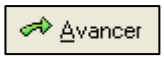
.....

.....

.....

.....

.....

- ✓ Créer un projet **`Prj_sp_eq_deg1.dev`** et associer un nouveau fichier à renommer **`main_resol_eq_deg1.c`** .
- ✓ Développer le sous-programme **`saisie_coeff`** dans le fichier **`main_resol_eq_deg1.c`** et le tester en mode pas à pas (**F7** 

) **après avoir traduit son appel dans le programme principal**.
- ✓ Compléter en langage algorithmique la procédure **resolution_deg1** permettant de resoudre l'équation.

Procédure resolution_deg1 (…… **)**

……

……

……

……

……

……

……

……

……

……

……

……

……

……

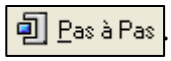
……

……

……

✓ Développer le sous-programme **`resolution_deg1`** dans le fichier **`main_resol_eq_deg1.c`** et le tester en mode pas à pas (**F7** 

) **après avoir traduit son appel dans le programme principal**.

Remarque : pour exécuter globalement la procédure saisie_coeff sans la déboguer, on utilisera la commande **F8** 

.

✓ Compléter en langage algorithmique la procédure **edite_solution** permettant d'éditer les différents cas de solution. Le résultat d'édition devra être conforme à celui présenté au TP N°1 aux pages 8-9.

Procédure edite_solution (……)

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

✓ Développer le sous-programme **`edite_solution`** dans le fichier **`main_resol_eq_deg1.c`** et le tester en mode pas à pas **après avoir traduit son appel dans le programme principal**.

TRAVAUX SUPPLEMENTAIRES

4. Développement d'une procédure puissance

On souhaite développer un sous-programme **puissance** permettant d'effectuer le calcul de la puissance entière $Y = X^p$ d'un nombre réel (Cf TP n°2) lorsque cela est possible :

✓ Compléter le programme principal ci-dessous écrit en langage algorithmique :

VAR

X : ..... /* variable de la fonction Y = X^P */

P : ..... /* puissance entière */

Y : ..... /* Résultat du calcul de la puissance X^P */

Calcul_valid : ..... /* Drapeau permettant d'indiquer si le calcul a pu être effectué */

Début

/* Saisie de la variable X et de la puissance entière */

Ecrire (" Calcul de Y = X^P ")

.....

.....

.....

.....

/* Traitement */

puissance(Y, X, P , Calcul_valid)

/* Edition */

.....

.....

.....

.....

.....

Fin

✓ Ecrire la procédure **puissance** en langage algorithmique en précisant la nature et le type des paramètres formels utilisés. Faire l'inventaire des variables locales nécessaires en reprenant l'algorithme déjà étudié à la page 22 du TP N°2.

Procédure puissance (……)

……

……

……

……

……

Variables locales :

……

……

……

Début

……

……

……

……

……

……

……

……

……

……

……

……

……

.....

.....

.....

.....

.....

.....

.....

.....

Fin

- ✓ Créer un projet **`Prj_puissance.dev`** et et associer un nouveau fichier à renommer **`main_puissance.c`**.
- ✓ Développer et tester le sous-programme **`puissance`** en mode pas à pas dans le fichier **`main_puissance.c`**.

5. Développement d'une procédure de développement limité de la fonction exp (*x*)

On reprend l'exercice 5 du TP N°2 (page 29) afin d'écrire une procédure **developpement_exp** permettant, connaissant la variable X et la précision **du calcul souhaitée** (en **%**) :

- le calcul de **exp(*x*)** à l'aide d'un polynôme,
- l'ordre **N** du polynôme pour obtenir la **précision souhaitée**.

✓ Compléter le programme principal ci-dessous écrit en langage algorithmique :

VAR

X : ..... /*Variable utilisée pour le calcul */

P_X : ..... /* Calcul du polynôme P (X) */

Prec_souhaitee : ..... /* Précision du développement souhaitée */

N : ..... /* Ordre du développement nécessaire à l'obtention de la précision souhaitée */

Prec_obtenue : ..... /* Précision réellement obtenue */

Début

/* Saisie de la variable X et de la précision souhaitée */

.....

.....

.....

.....

/* Calcul du développement */

developpement_exp (X, Prec_souhaitee, P_X, N, Prec_obtenue)

/* Edition des résultats : P_X, N, Prec_obtenue et comparaison de P_X avec le calcul direct de la fonction exp(X) de la bibliothèque <math.h> de devcpp*/

.....

.....

.....

.....

.....

.....

✓ Ecrire la procédure **developpement_exp** en langage algorithmique en précisant le nombre, la nature et le type des paramètres formels utilisés.

Procédure developpement_exp (……)

……

……

……

……

……

……

- ✓ Créer un projet **`Prj_DL_exp_SP.dev`** et associer un nouveau fichier à renommer **`main_exp_SP.c`** .
- ✓ Développer le sous-programme **`developpement_exp`** en langage C.

Un exemple d'écran d'exécution attendu est fourni ci-dessous :

```
Entrer la valeur de x : 1
Entrer la precision desiree : (en pourcent) 0.1

exp ( 1.00) peut etre approche par un polynome d'ordre 6
avec une precision relative +/- (5.11E-002) %


exp ( 1.00) = 2.7182818E+000


P( 1.00) = 2.7180557E+000

Appuyez sur une touche pour continuer... _
```

SOUS-PROGRAMMES : MANIPULATION DE TABLEAUX

Thèmes abordés : *Tableaux et pointeurs (fascicule* **Eléments de base du langage C***, p.18) Passage de paramètres tableaux de nombres dans des procédures ou fonctions (fascicule* **Eléments de base du langage C***, p.22-23).*

1. Pointeurs et tableaux à une dimension

Ouvrir le projet **`Prj_pointeurs_tab.dev`** situé dans votre répertoire **`U:\Prog_C\TP6`** et éditer le fichier **`main_pointeurs_tab.c`** dont le listing est fourni ci-dessous :

```
/* Nom de fichier disque : main_pointeurs_tab.c
   Projet : Prj_pointeurs_tab.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : TP N°6 : Pointeurs et tableaux
   Date :
*/
#include <stdio.h>
#include <stdlib.h>
#define N_MAX 5

typedef float t_reels [N_MAX] ;

int main()
{
   // Variables du programme principal
   t_reels Liste = {0,1,2,3,4} ;
   int i         ;
   float *t      ; // pointeur de réel (float)

   // Corps exécutable du programme principal

   /* Manipulation du contenu de liste à l'aide
      de la variable pointeur t */

   t = Liste ;
   for(i=0; i< N_MAX; i++)
   {
      t[i] = i+0.5 ;
   }
   // Edition des valeurs de liste

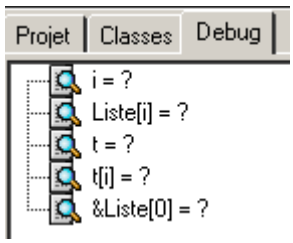
   for (i=0; i < N_MAX; i++)
   {
       printf("Liste [%d] = %6.2f\n",i,Liste[i]) ;
   }
   printf("\n") ;
   system("PAUSE");
   return 0;
}
```

Expliquer quel est le type précis de la variable Liste ?

.....

.....

✓ Tester le programme en mode pas à pas à l'aide du débogueur.



Contrôler en particulier le contenu des variables **`Liste[i]`**, **`t`**, **`t[i]`**. En déduire l'adresse du premier élément du tableau **`Liste`**.

Si il y a un problème sur la lecture de t[i], activer à nouveau la commande Ajout Variable .

`&Liste[ 0 ] =`	**`t =`** .....
`Liste [ 0 ] =`	**`t [ 0 ] =`** .....
`Liste [ 1 ] =`	**`t [ 1 ] =`** .....
`Liste [ 2 ] =`	**`t [ 2 ] =`** .....
`Liste [ 3 ] =`	**`t [ 3 ] =`** .....
`Liste [ 4 ] =`	**`t [ 4 ] =`** .....

2. Procédure de lecture des valeurs significatives d'un tableau de nombres

✓ Compléter en langage algorithmique la procédure **`lit_tableau`** permettant de lire **n** valeurs significatives d'une variable tableau **t** de nombre réels pouvant contenir au plus **capacite_stockage** éléments. L'appel de ce sous-programme pourra être de la forme :

lit_tableau (Liste , N, N_MAX)

Dans la procédure, **n** devra être saisi avec la garantie que **1 ≤ n ≤ capacite_stokage**.

L'acquisition des valeurs au clavier pourra se faire par l'intermédiaire d'une variable locale **aux** *avant l'affectation dans le tableau.*

Procédure lit_tableau (t , n, capacite_stockage)

.....

.....

.....

.....

.....

.....

.....

.....

.....

✓ Ouvrir le projet **`Prj_sp_tab.dev`**, développer la procédure **`lit_tableau`** dans le fichier **`main_sp_tab.c`** et insérer l'appel à ce sous-programme dans le programme principal (on testera ce sous-programme en mode pas à pas).

Un exemple d'écran d'exécution attendu est fourni ci-dessous :

```
Entrer une valeur comprise entre 1 et 10 : 15
Recommencer, il n'y a pas assez de place disponible : 3

Saisie de la 1ere valeur : -15.5
Saisie de la 2eme valeur : 12.5
Saisie de la 3eme valeur : -7


Appuyer sur une touche pour continuer . . .
```

3. Procédure d'affichage des valeurs significatives d'un tableau de nombres

✓ Compléter en langage algorithmique la procédure **`affiche_tableau`** permettant d'éditer les **n** valeurs significatives d'une variable tableau **t** de nombre réels.

Procédure affiche_tableau (t , n)

✓ Développer la procédure **`affiche_tableau`** dans le fichier **`main_sp_tab.c`** et insérer l'appel à ce sous-programme dans le programme principal (on utilisera ce sous-programme pour afficher le contenu de la variable **`Liste`**).

Un résultat d'exécution pourra être :

```
Liste des valeurs significatives de Liste :

 1ere valeur:-15.50, 2eme valeur: 12.50, 3eme valeur: -7.00,

Appuyer sur une touche pour continuer . . .
```

4. Fonction min_ou_max

✓ Ecrire en langage algorithmique une fonction **min_ou_max** permettant de renvoyer la plus petite ou la plus grande des valeurs parmi les **n** valeurs significatives rangées dans une variable tableau **t** de nombres **réels**.

L'appel dans le programme principal s'écrit :

Liste_min ← **min_ou_max** (**Liste** , **N** , **'-'**)

Liste_max ← **min_ou_max** (**Liste** , **N** , **'+'**)

où **Liste_min** et **Liste_max** désignent des variables du programme principal de type réel permettant de stocker la plus petite valeur et la plus grande valeurs parmi les **N** valeurs significatives du tableau **Liste**.

Fonction min_ou_max **(t , n , choix) :** .....

.....

.....

.....

✓ Développer la fonction **`min_ou_max`** dans le fichier **`main_sp_tab.c`**.
Un résultat d'exécution pourra être :

```
Liste des valeurs significatives de Liste :

 1ere valeur:  5.00, 2eme valeur:  7.00, 3eme valeur:  8.00, 4eme valeur:  4.00,
 5eme valeur:  6.00,
La valeur minimale parmi les 5 valeurs contenues dans Liste est 4.00
La valeur maximale parmi les 5 valeurs contenues dans Liste est 8.00

Appuyer sur une touche pour continuer . . .
```

5. Fonction select_negatifs

✓ Ecrire en langage algorithmique une fonction **select_ negatifs** permettant de **renvoyer** le nombre de valeurs négatives ou nulles parmi les valeurs significatives rangées dans un tableau de nombres réels. De plus, cette fonction devra permettre de récupérer dans le programme principal un tableau contenant ces valeurs.

L'appel dans le programme principal s'écrit :

N_neg ← **select_negatifs** (**Liste** , **N** , **Liste_neg**)

où **Liste** désigne l'ensemble des valeurs saisies au clavier, **Liste_neg** la variable permettant de récupérer les **N_neg** valeurs négatives ou nulles contenues dans **Liste**.

Fonction select_negatifs **(t , n , t_neg) :**

.....

.....

.....

.....

.....

.....

.....

.....

.....

TRAVAUX SUPPLEMENTAIRES

6. Conception d'un jeu de réflexion – jeu des permutations

6.1. Principe du jeu

On considère la liste des **10** premiers chiffres **entiers**, rangée dans une variable tableau **liste_jeu** d'au plus **10 entiers** :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

L'ordinateur commence par mélanger l'ordre des chiffres de la variable **liste_jeu** de manière **aléatoire** et affiche le contenu des chiffres ainsi mélangés, *par exemple* **:**

3, 4, 9, 0 , 6, 8, 2, 1, 5 , 7

Le jeu consiste alors à reclasser la liste initiale dans l'ordre croissant avec les règles suivantes :

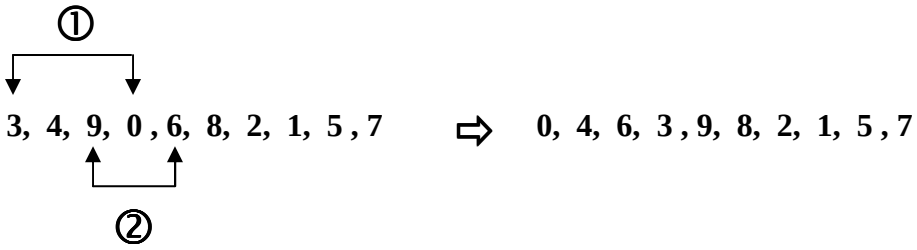
A chaque étape, l'utilisateur choisit le chiffre qu'il désire placer en première position, ce qui entraîne les modifications ci-après :

① le chiffre choisi est permuté avec le premier,

② les 2 chiffres collatéraux gauche et droite au chiffre choisi sont ensuite permutés.

Exemple : **3, 4, 9, 0 , 6, 8, 2, 1, 5 , 7**

Si l'utilisateur choisi de placer le chiffre **0** au début, on obtient la modification suivante :



Cas particuliers des bords extrêmes :

Si l'utilisateur choisit de placer en premier le premier chiffre, le deuxième et le dixième sont permutés :

Exemple : **3, 4, 9, 0 , 6, 8, 2, 1, 5 , 7**

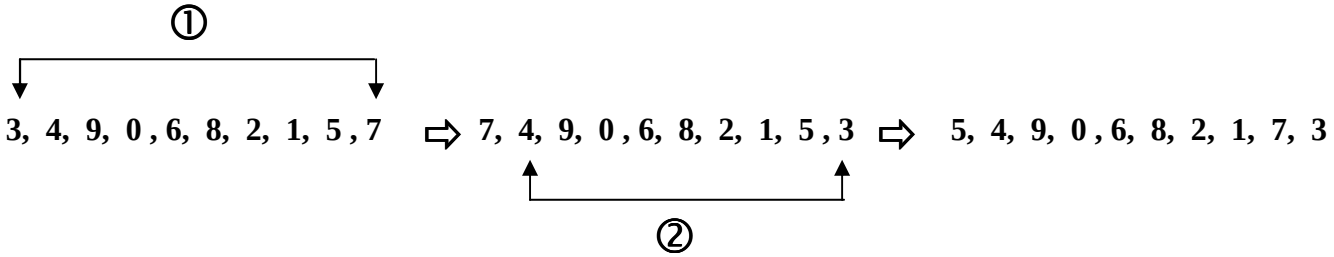
Si l'utilisateur choisi de placer le chiffre **3** au début (il y est déjà !), on obtient la modification suivante :



Si l'utilisateur choisit de placer en premier le dernier (dixième) chiffre, le neuvième chiffre est placé en première position et le premier en dixième position.

Exemple : **3, 4, 9, 0 , 6, 8, 2, 1, 5 , 7**

Si l'utilisateur choisi de placer le chiffre **7** au début, on obtient la modification suivante :



On peut réduire la difficulté du jeu en limitant le classement des **nb_chiffres** premiers chiffres compris entre **0** et **nb_chiffres–1** avec **nb_chiffres ≤ 10**.

On autorise le joueur à un nombre maximal d'essai **NB_MAX_ESSAIS** pour reclasser la liste initiale.

Ouvrir le fichier projet **Prj_jeu_permute.dev** et éditer le fichier **main_jeu_permute.c** dans lequel on écrira les différents sous-programmes demandés.

```
/*
  Nom de fichier disque : main_jeu_permute.c
  Projet : Prj_jeu_permute.dev
  Prénom :
  Nom    :
  Groupe :
  Commentaire :
  Date :
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N_MAX 10
#define NB_MAX_ESSAIS 25 // nombre d'essais maximum pour reclasser
                         // la liste initiale
void randomize (void) ; // Initialise le générateur de nombre
                        // aléatoire
int random (int n)    ; // renvoie un nombre aléatoire compris
                        // entre 0 et n-1
int main()
{
  // Déclaration des variables du programme principal

  // Déclarer une variable Liste_jeu d'au plus 10 valeurs entières
  // initialisées avec les valeurs 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    printf("\n\n") ;
    system("PAUSE");
    return 0;
}
```

```
void randomize (void)
{
  srand( (unsigned)time( NULL ) ); // initialise le générateur de
                                   // nombre aléatoire
}
int random (int n)
{
   int v_alea ;

   v_alea = rand() % n ;
   return v_alea ;
}
```

Des exemples d'exécution sont fournis sur les écrans ci-dessous. Il est possible de faire fonctionner le programme demandé en exécutant le fichier **Prj_jeu_permute_soluce.exe**.

```
Combien de chiffres souhaitez-vous classer ? (entre 3 et 10) : 4

Voici la liste initiale a ranger dans l'ordre croissant :

3 0 2 1             1er essai : chiffre a placer au debut ? 2
2 1 3 0            2eme essai : chiffre a placer au debut ? 1
3 2 1 0            3eme essai : chiffre a placer au debut ? 0
1 2 0 3            4eme essai : chiffre a placer au debut ? 2
0 1 2 3            Bingo !

Vous avez gagne  en 4 essais !

Appuyer sur une touche pour continuer ...
```

```
Combien de chiffres souhaitez-vous classer ? (entre 3 et 10) : 5

Voici la liste initiale a ranger dans l'ordre croissant :

1 0 3 4 2            1er essai : chiffre a placer au debut ? 1
1 2 3 4 0            2eme essai : chiffre a placer au debut ? 0
4 2 3 0 1            3eme essai : chiffre a placer au debut ? 5
Erreur de saisie, recommencer : 1
0 2 3 1 4            4eme essai : chiffre a placer au debut ? 4
1 2 3 4 0            5eme essai : chiffre a placer au debut ? 3
3 4 1 2 0            6eme essai : chiffre a placer au debut ? 3
3 0 1 2 4            7eme essai : chiffre a placer au debut ? 1
1 2 3 0 4            8eme essai : chiffre a placer au debut ? 0
0 2 4 1 3            9eme essai : chiffre a placer au debut ? 0
0 3 4 1 2           10eme essai : chiffre a placer au debut ? 1
1 3 2 0 4           Stop !

Vous n'avez pas pu trier la liste en moins de 10 essais

Appuyer sur une touche pour continuer ...
```

6.2. Programmation

6.2.1. Fonction lit_entier_dans_intervalle

Cette fonction sera utilisée plusieurs fois dans ce projet. Elle doit permettre de retourner une valeur entière lue au clavier comprise entre deux bornes définissant un intervalle.

Ainsi, l'appel

nb_chiffres ← lit_entier_dans_intervalle (3, 10)

doit permettre de lire une valeur entière comprise dans l'intervalle **[3 ; 10]**.

Fonction lit_entier_dans_intervalle (.....) : .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

- ✓ Développer cette fonction en C tester la.

6.2.2. Procédure affiche_liste

- ✓ Ecrire en langage algorithmique une procédure **affiche_liste** permettant d'afficher **sur 1 ligne** la liste des **n** valeurs significatives contenues dans un tableau **t** d'au plus **N_MAX** entiers.

Procédure affiche_liste (……)

……

……

……

……

……

……

……

……

……

……

✓ Après avoir initialisé les valeurs de **`Liste_jeu`**, développer ce sous-programme en C tester le pour afficher les **10** valeurs contenues dans **`Liste_jeu`**.

6.2.3. Procédure melange

✓ Ecrire en langage algorithmique un sous_programme **melange** permettant de redistribuer les **n** différentes valeurs entières valeurs contenues dans un tableau **t** d'au plus **N_MAX** entiers et dont l'organigramme est fourni ci-dessous.

Procédure melange (t , n)

……

……

……

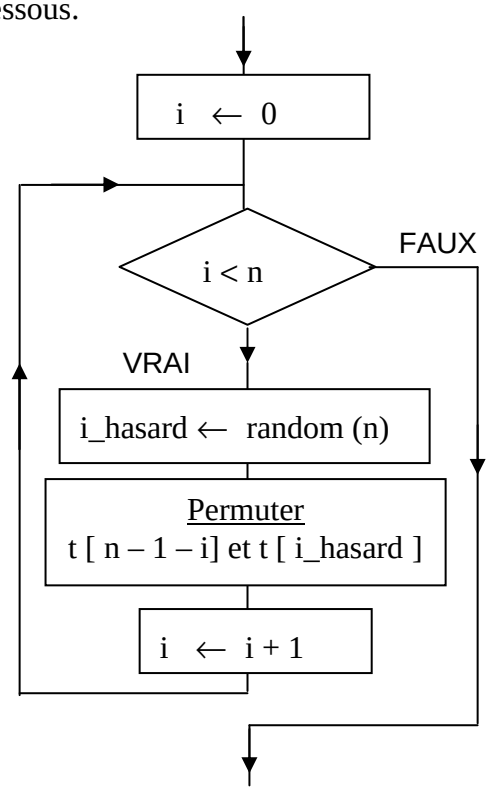
……

……

……

……

……



.....
.....
.....
.....
.....
.....

On suppose que la fonction **random (n)** renvoie un nombre entier compris entre **0** et **n – 1**.

✓ Développer ce sous-programme en C tester le en mode pas à pas.

6.2.4. Procédure choix_1er_element

✓ Ecrire une procédure **choix_1er_element** permettant de modifier le tableau **liste_jeu** contenant les n premiers chiffres entiers selon la règle énoncée dans le principe du jeu (§.1)

L'organigramme du corps exécutable de la procédure est fournie sur l'organigramme à la page suivante.

Procédure choix_1er_element (**t** , **n**)
.....
.....
.....

Variables locales
.....
.....
.....
Début
.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

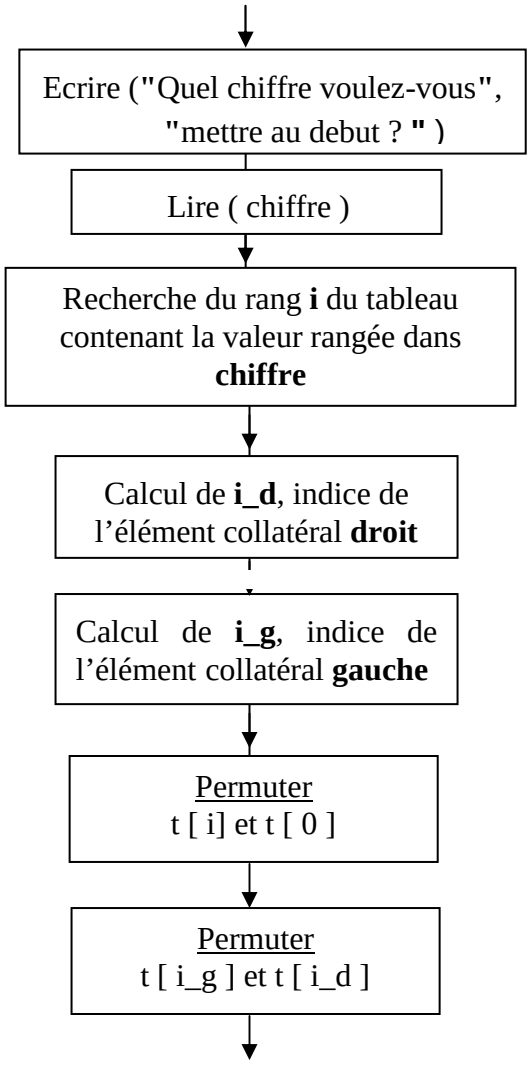
.....

.....

.....

.....

Fin

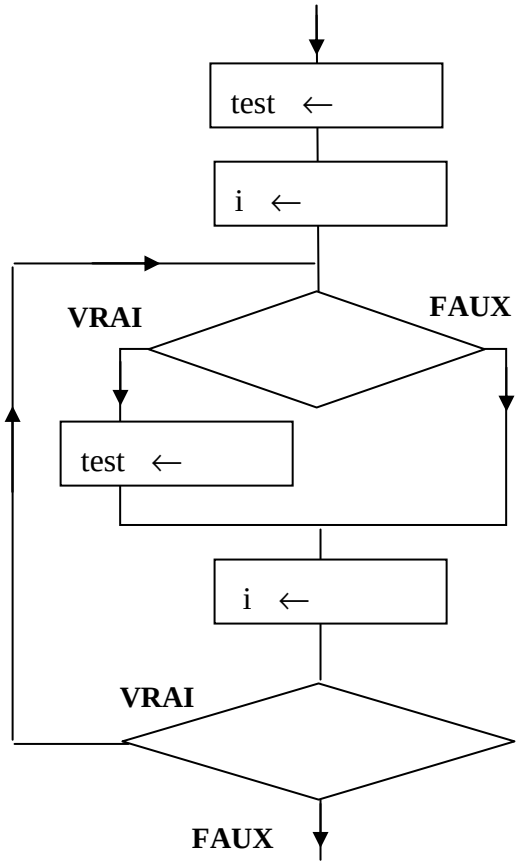

Ecrire ("Quel chiffre voulez-vous", "mettre au debut ? ")
Lire (chiffre)
Recherche du rang **i** du tableau contenant la valeur rangée dans **chiffre**
Calcul de **i_d**, indice de l'élément collatéral **droit**
Calcul de **i_g**, indice de l'élément collatéral **gauche**
Permuter t [i] et t [0]
Permuter t [i_g] et t [i_d]

6.2.5. Fonction test_liste_ordonnee

✓ Ecrire en langage algorithmique une fonction **test_liste_ordonnee** permettant de retourner la valeur **1** dans le cas où les **n** différentes valeurs entières valeurs contenues dans un tableau **t** d'au plus **N_MAX** entiers sont rangées dans l'ordre **0, 1, 2, 3, …, n** et **1** dans le cas contraire.

Fonction test_liste_ordonnee (t, n) :

L'organigramme du corps exécutable est fourni par l'organigramme ci-dessous :



✓ Développer et tester cette fonction en C.

PROGRAMME DE CRYPTAGE DE CHAINES DE CARACTERES : LE CODE DE CESAR

Thèmes abordés : *caractères, chaînes de caractères*

1. Le code de césar



Le code de César est la méthode de cryptographie la plus ancienne communément admise par l'Histoire. Il consiste en une substitution mono-alphabétique, où la substitution est définie par un décalage de lettres (**écrites en majuscule**). Par exemple, si on remplace **A** par **D**, on remplace **B** par **E**, **C** par **F**, **D** par **G**, etc...

L'exemple suivant correspond à un décalage de **3 lettres** :

Caractère clair	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
Caractère codé	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**

Attention : ce décalage devra se faire selon une permutation circulaire.

Remarque : il n'y a que **25 façons différentes** de crypter un message avec le code de César. Cela en fait donc un code très peu sûr, puisqu'il est très facile de tester de façon exhaustive toutes les possibilités. Pourtant, en raison de sa grande simplicité, le code de César fut encore employé par les officiers sudistes pendant la Guerre de sécession, et même par l'armée russe en 1915.

Les programmes demandés devront permettre d'élaborer des procédures de codage et de décodage selon le code de César sur des chaînes de caractères.

2. Principe du codage d'une lettre majuscule (fonction decalage)

Les lettres majuscules ont un code ASCII de valeur décimale comprise entre **65** et **90**. La majuscule à coder est stockée dans la variable **Lettre_a_coder** qui sera transmise au paramètre formel **lettre** de la fonction **decalage** selon le code de César.

La valeur du décalage, comprise entre **1** et **25**, est stockée dans la variable **Code** (transmise au paramètre formel **clef**) et sera appliqué au rang alphabétique (variable **rang_alpha**) afin d'en déduire le caractère après codage.

✓ En consultant le tableau ci-dessous, quelle instruction en langage algorithmique permet d'obtenir, à partir du code ASCII, la valeur de son rang alphabétique.

.....

Caractère	**'A'**	**'B'**	**'C'**	**'D'**	.....	**'W'**	**'X'**	**'Y'**	**'Z'**
Code ASCII	**65**	**66**	**67**	**68**	.....	**87**	**88**	**89**	**90**
rang_alpha	**1**	**2**	**3**	**4**	.....	**23**	**24**	**25**	**26**

✓ En supposant que **clef** vaut **3** (décalage de 3 lettres), déterminer la valeur du rang alphabétique (variable **rang_alpha_code**) du caractère après décalage, en déduire les caractères codés correspondants.

rang_alpha_code					.....				
Caractère codé	**'D'**				.....				**'C'**

✓ En utilisant l'opérateur **mod** (reste de la division entière), déterminer l'instruction permettant de calculer la valeur du rang alphabétique codé à partir de la valeur du rang alphabétique contenue dans **rang_alpha** et de la valeur de décalage contenue dans **clef** :

rang_alpha_code ← .....

✓ Vérifier que l'opérateur **mod** permet de réaliser la permutation circulaire.

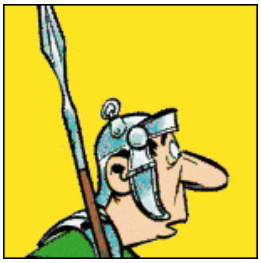
✓ Peut-on avoir **rang_alpha_code** = 26 avec l'instruction précédente ? ……

Prévoir le traitement nécessaire pour le calcul correct de **rang_alpha_code** de l'ensemble des caractères :

.....

.....

✓ Compléter l'organigramme du traitement de la fonction **decalage** permettant de coder un caractère contenu dans **lettre** avec un décalage **à droite** d'une valeur contenue dans **clef**. Le résultat sera stocké dans **lettre_codee** et retourné par la fonction.



✓ Ecrire en langage algorithmique la fonction **decalage**.

L'appel dans le programme principal pourra s'écrire :

Lettre_codee ← **decalage** (**Lettre_a_coder**, **Code**)

Fonction decalage (lettre, clef) : .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

✓ Ouvrir le projet **Prj_cesar.dev** (dans le répertoire **U:\Prog_C\TP7**) et le fichier **main_cesar.c** dont le listing est fourni ci-après :

```
/*
   Nom de fichier disque : main_cesar.c
   Projet : Prj_cesar.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : TP N°7 : LE CODE DE CESAR
   Date :
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NB_CAR_MAX 80

typedef char t_car[NB_CAR_MAX] ;

int main()
{
  // Déclaration des variables du programme principal

   t_car Phrase = "JE SUIS ETUDIANT EN G.E.I.I." ;
   t_car Phrase_codee ;
   char  Lettre_a_coder, Lettre_codee ;
   int   Code = 10 ;    // Code de cryptage
// Corps exécutable du programme principal

// Copie de la chaîne de caractères contenue dans Phrase dans le
// tableau Phrase_codee
  strcpy (Phrase_codee, Phrase) ;

// 1ere fonction à tester
// Lettre_codee = decalage (Lettre_a_coder, Code) ;
// Lettre_a_coder = 'A', Code = 3 ==> Lettre_codee = 'D'
// Lettre_a_coder = 'Y', Code = 3 ==> Lettre_codee = 'B'
// Lettre_a_coder = '?', Code = 3 ==> Lettre_codee = '?'

// 2e sous-programme à tester :
// code_cesar (Phrase_codee, Code) ;
   printf("Phrase en clair : %s\n",Phrase) ;
   printf("Phrase codee    : %s\n",Phrase_codee) ;
   printf("\n") ;
   system("PAUSE");
   return 0;
}
```

✓ Quel est **précisément le type** des variables **Phrase** et **Phrase_codee** ?

.....

✓ Combien de caractères peut contenir la chaîne de caractères mémorisée dans ces variables (justifier votre réponse) ?

.....

✓ En exécutant le programme en mode pas à pas jusqu'à la première ligne du bloc exécutable et en consultant le contenu de la variable **`Phrase`** préciser les valeurs des caractères suivants ainsi que leur code **ASCII** correspondant (en décimal) :

`Phrase [ 0 ]` **=** .....

`Phrase [ 9 ]` **=** .....

`Phrase [ 27 ]` **=** .....

`Phrase [ 28 ]` **=** .....

Que peut-on dire de l'effet de l'initialisation de la variable **`Phrase`** sur les caractères placés après **`Phrase [ 28 ]`** ?

.....

✓ Développer en C la fonction **`decalage`** et la tester en mode pas à pas. On vérifiera le résultat de l'instruction **`Lettre_codee`** `= ` **`decalage`** `(Lettre_a_coder,Code);` pour les valeurs `'A'`, `'Y'`, puis `'?'` avec `Code = 3`.

Une fois testée, mettre l'appel à la fonction `decalage` **à nouveau en commentaires**

3. Codage d'une chaîne de caractères

✓ En utilisant la fonction **decalage**, écrire une procédure **code_cesar** permettant de coder une chaîne de caractères, connaissant la valeur du code de décalage.

L'appel s'écrit : **code_cesar (Phrase_codee, Code)**

Procédure code_cesar (s , clef)

Paramètres formels :

s : .....

clef : .....

Variable locale :

.....

.....

.....

.....

.....

.....

.....

.....

✓ Développer et tester ce sous-programme en C.

Le résultat d'exécution pourra être de la forme (pour une valeur de **Code** initialisée à **10**) :

```
Phrase en clair : JE SUIS ETUDIANT EN G.E.I.I.
Phrase codee    : TO CESC ODENSKXD OX Q.O.S.S.
Appuyer sur une touche pour continuer . . .
```

4. Détermination du code d'un message codé

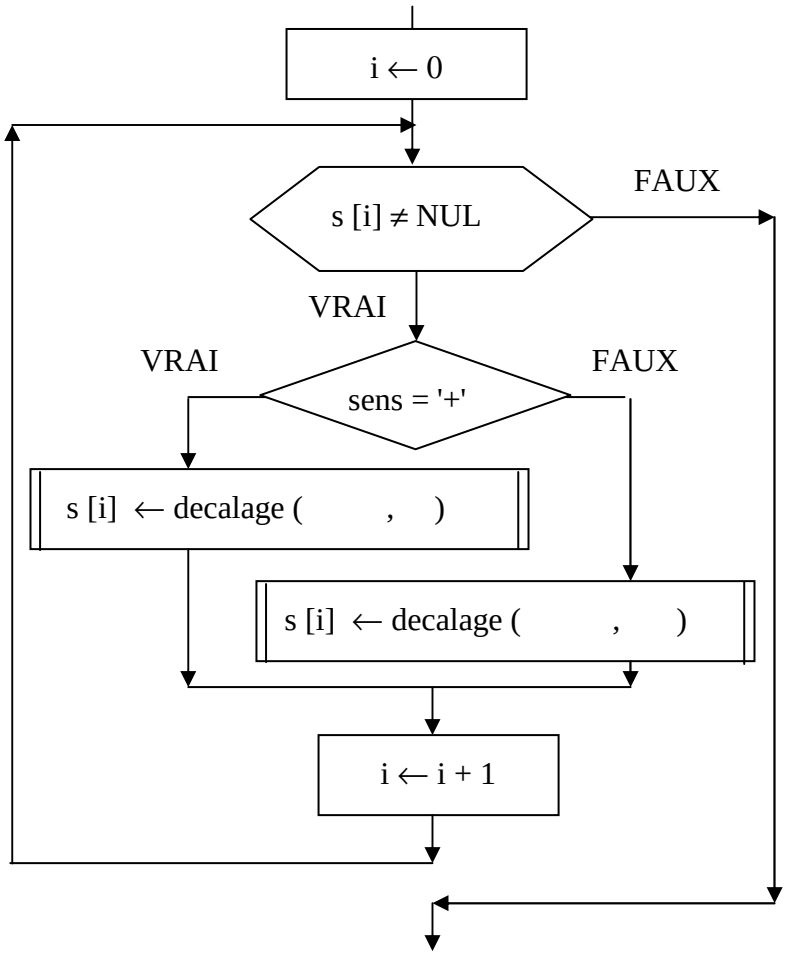
4.1. Décalage des caractères à gauche ou à droite d'un message

La procédure **decal_message** doit permettre le décalage de l'ensemble des caractères d'une chaîne **d'un caractère** à gauche ou à droite. Le sens du décalage est fonction de la valeur '**+**' ou '**-**' transmise au paramètre formel **sens**.

Supposons que Phrase contienne initialement la chaîne **"I.U.T.".**

Après l'appel	Après l'appel
decal_message (Phrase, '**+**')	**decal_message (**Phrase, '**-**')
Phrase devra contenir **"J.V.U."**	Phrase devra contenir **"H.T.S."**

✓ Compléter l'organigramme ci-dessous de traitement de la procédure



✓ Ecrire en langage algorithmique la procédure **decal_message**.

Procédure decal_ message (s, sens)

Paramètres formels :

s : t_car { .....}

sens : caractère { .....}

Variable locale :

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

✓ Ouvrir le projet **Prj_casse_code.dev** (dans le répertoire **U:\Prog_C\TP7**) et le fichier **main_casse_code.c** dont le listing est fourni ci-après :

```
/*
   Nom de fichier disque : main_casse_code.c
   Projet : Prj_casse_code.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire : TP N°7 : Recherche de la clef de codage d'un message
                          crypté selon le code de CESAR
   Date :
*/

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

#define NB_CAR_MAX 80

typedef char t_car[NB_CAR_MAX] ;
```

```
int main()
{
   // Déclaration des variables du programme principal
   // Phrase_codee et Phrase_codee_bis ont été cryptées avec la même
   // clef de codage

   t_car Phrase_codee = "AP KPATJG SJ RDST THI BDXCH FJXCOT DJ DCOT" ;
   t_car Phrase_codee_bis = "KDJH TITH TBQPJRWT BDCHXTJG QDCS !" ;
   t_car Phrase ="I.U.T." ;
   int Code_a_trouver  ;    // Code de cryptage

   // Corps exécutable du programme principal

// decal_message (Phrase, '+') ;  // test du SP decal_message
// decal_message (Phrase, '-') ;

// 1ère étape : recherche du code

// Copie de la chaîne de caractères contenue dans Phrase_codee dans le
// tableau Phrase_decodee
//  strcpy (Phrase_decodee, Phrase_codee) ;
// recherche_code (Phrase_decodee, &Code_a_trouver) ;

// Edition des résultats :

   system("PAUSE");
   return 0;
}
```

✓ Après avoir recopié le prototype et la définition de la fonction **`decalage`** écrits dans le fichier **`main_cesar.c`**, tester en mode pas à pas le sous-programme **`decal_message`**.

4.2. Recherche du code d'un message codé (procedure recherche_code)

Le principe est d'afficher à l'écran le résultat des décalages successifs d'une unité à gauche ou à droite d'un message codé par des appuis successifs sur la touche '**-**' ou '**+**' du clavier. Lorsque le message devient lisible, on arrête la recherche en appuyant sur la touche '**x**'.

Pour effectuer le codage, on a procédé à un décalage des caractères dans un sens, par conséquent, pour décoder, il faut effectuer le décalage dans l'autre sens. Cette opération est réalisée en transmettant le caractère '**-**' au paramètre sens de la procédure **decal_message**.

Lorsque la phrase est écrite en clair, le paramètre **clef** contient alors la valeur du code.

Remarque : l'organigramme proposé permet également de traiter le cas où on utilise majoritairement la touche '**+**' pour décoder le message. Expliquer pourquoi dans ce cas le code à trouver correspond à la valeur **26** + **clef**.

……

……

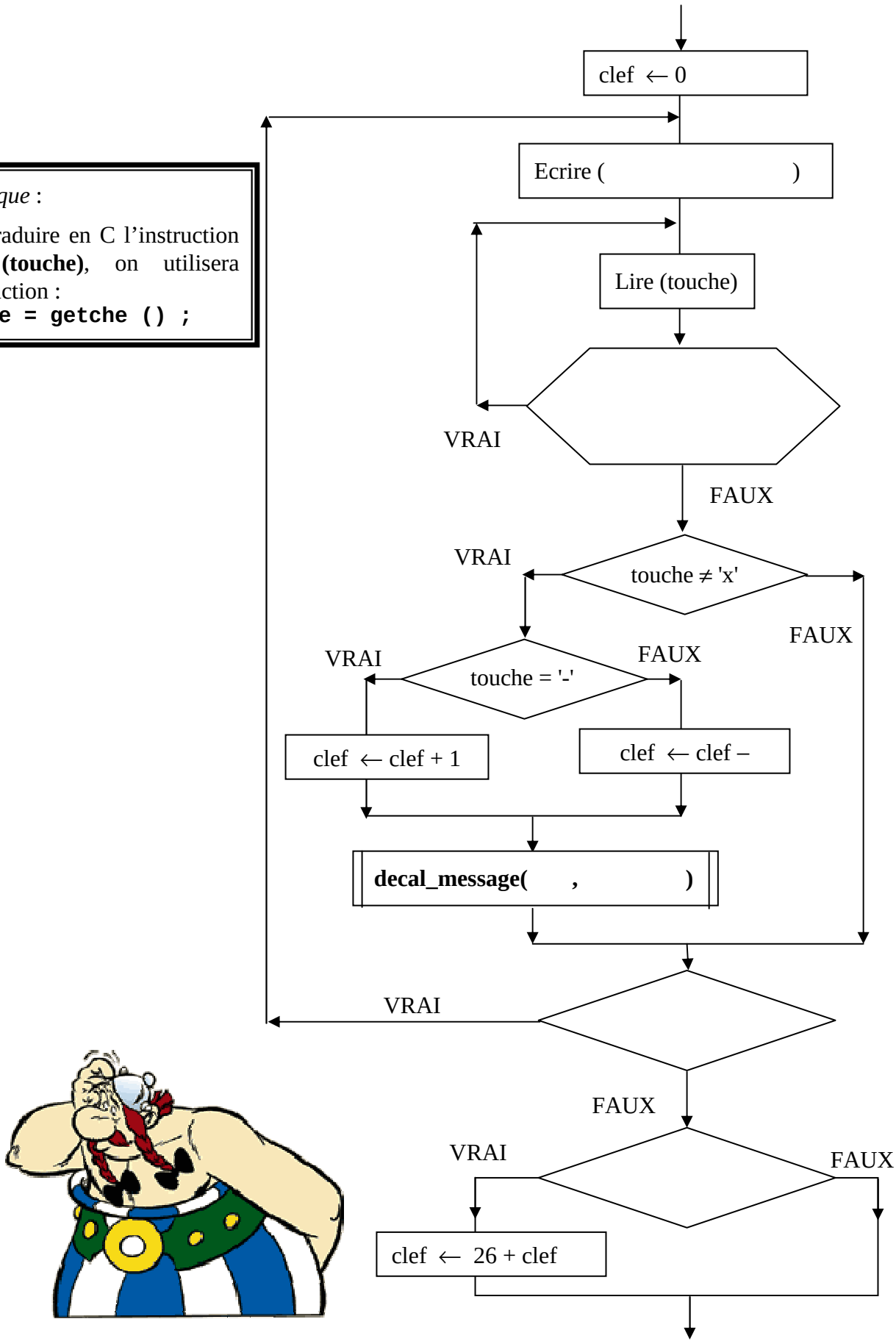
……

……..

Remarque :

Pour traduire en C l'instruction **Lire (touche)**, on utilisera l'instruction :

```
touche = getche () ;
```



✓ Ecrire en langage algorithmique le sous-programme **recherche_code** permettant de fournir au programme principal le message décodé ainsi que le code correspondant.

L'appel de ce sous-programme s'écrit en langage algorithmique :

recherche_code (Phrase_decodee, Code_a_trouver)

Procédure recherche_code (s, clef)

Paramètres formels :

s : t_car {.....}

clef : Entier {.....}

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

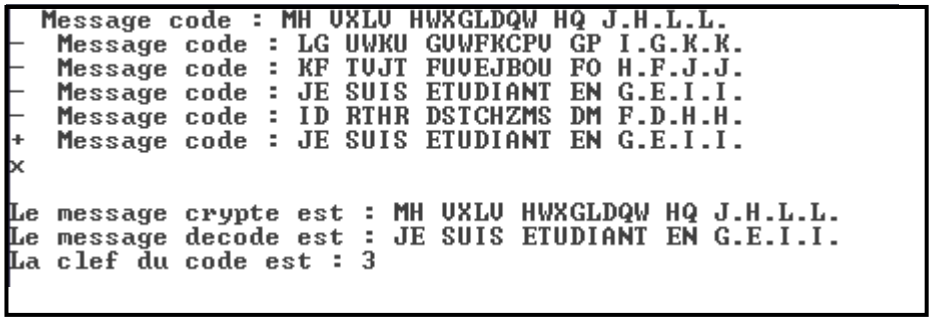
.....

.....

.....

✓ Développer le sous-programme **recherche_code** en C et fournir le code ainsi que le message en clair **Phrase_decodee** initialement stocké de manière cryptée dans **Phrase_codee**.

Un exemple d'exécution du programme est fourni sur l'écran ci-dessous :



4.3. Décodage direct d'un message avec le code connu

Connaissant la valeur du code (stockée dans **Code_a_trouver**) qui a servi au codage de la chaîne stockée dans **Phrase_codee**, sachant que le 2ème message stocké dans **Phrase_codee_bis** a été codé avec le même code, décoder directement le 2ème message en utilisant la procédure **code_cesar** (en déterminant la valeur à transmettre au paramètre **clef**).

Justification :

…..

……

……

……

Programme de tri à bulle (bubble sort)

On souhaite développer un algorithme de tri par **valeurs croissantes** en utilisant la technique du tri à bulle.

1. Première étape : recherche de la valeur minimale

La première étape consiste à écrire une procédure **deplace_min** permettant de **modifier** les valeurs du tableau en permutant **la plus petite valeur** avec la première.

Procédure deplace_min (t , n)

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

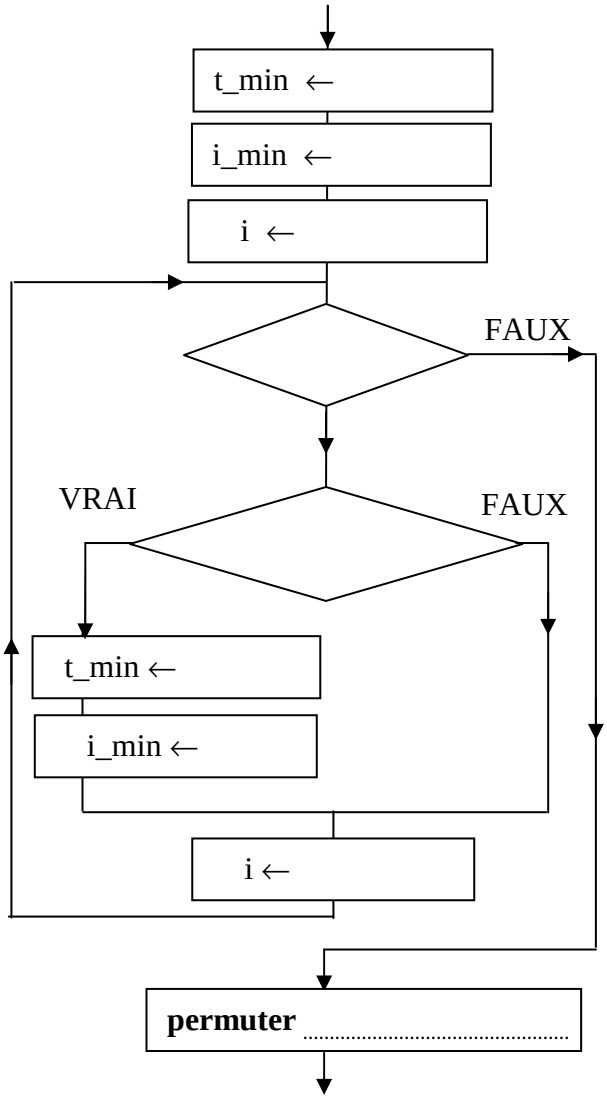
.....

.....

.....

.....

Ecrire en langage algorithmique le sous-programme **deplace_min**. On complètera l'organigramme du traitement correspondant fourni ci-dessous :



Exemple : on suppose que le contenu du tableau est initialement le suivant :

i	0	1	2	3	4
liste [i]	5.0	7.0	8.0	4.0	6.0

Après l'appel : **deplace_min** (**liste**, **N**), le tableau **liste** est transformé tel que :

i	0	1	2	3	4
liste [i]	4.0	7.0	8.0	5.0	6.0

✓ Ouvrir le projet **Prj_tri.dev** (dans le répertoire **U:\Prog_C\TP6**) et éditer le fichier **main_tri_num.c** dont le listing est fourni ci-après :

```
/*
   Nom de fichier disque : main_tri_num.c
   Projet : Prj_tri.dev
   Prénom :
   Nom    :
   Groupe :
   Commentaire :
   Date :
*/

#include <stdio.h>
#include <stdlib.h>

#define N_MAX 5
typedef float t_reels [N_MAX] ;

int main()
{
   // Variables du programme principal

   t_reels Liste = {5.0, 7.0, 8.0, 6.0, 4.0 } ;
   int N = 5 ; /* Nombre de valeurs significatives du tableau
                  liste */
    t_reels Liste_triee ;

   // Corps exécutable du programme principal

   copie_tableau (Liste_triee, liste, N) ;
// deplace_min () ;
// tri_bulle() ;

   printf("\n") ;
   system("PAUSE");
   return 0;
}
```

✓ Recopier dans le fichier **main_tri_num.c**, à partir du fichier **main_sp_tab.c**, les lignes de codes du sous-programme **copie_tableau**.

✓ Développer le sous-programme **deplace_min** et le tester en mode pas à pas.

2. Deuxième étape : tri à bulle par ordre croissant

L'algorithme du tri bulle - ou bubble sort - consiste à regarder les différentes valeurs adjacentes d'un tableau **t**, et à les permuter si le premier des deux éléments est supérieur au second. L'algorithme consiste à faire descendre les plus petites valeurs au début du tableau. Ainsi, à l'étape **i**, on commence à comparer **t [i+1]** et **t [i]**. Si **t [i+1] < t [i]**, on effectue la permutation. On effectue ensuite les comparaisons des valeurs adjacentes **en remontant vers le début du tableau** tant que la valeur **t [i+1]** repérée à l'étape **i** reste plus plus petite que les éléments comparés.

Cet algorithme porte le nom de tri bulle car, petit à petit, les plus grands éléments du tableau remontent, par le jeu des permutations, en fin de tableau. Dans un aquarium il en va de même : les plus grosses bulles remontent plus rapidement à la surface que les petites qui restent collés au fond.

Si **Liste_triee** contient initialement des valeurs dans le désordre, l'appel permettant le tri dans le programme principal s'écrit : **tri_bulle (Liste_triee , N)** Ecrire en langage algorithmique la procédure **tri_bulle** donné sous forme d'organigramme ci-dessous :

Procédure tri_bulle (t , n)

Paramètres formels : .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

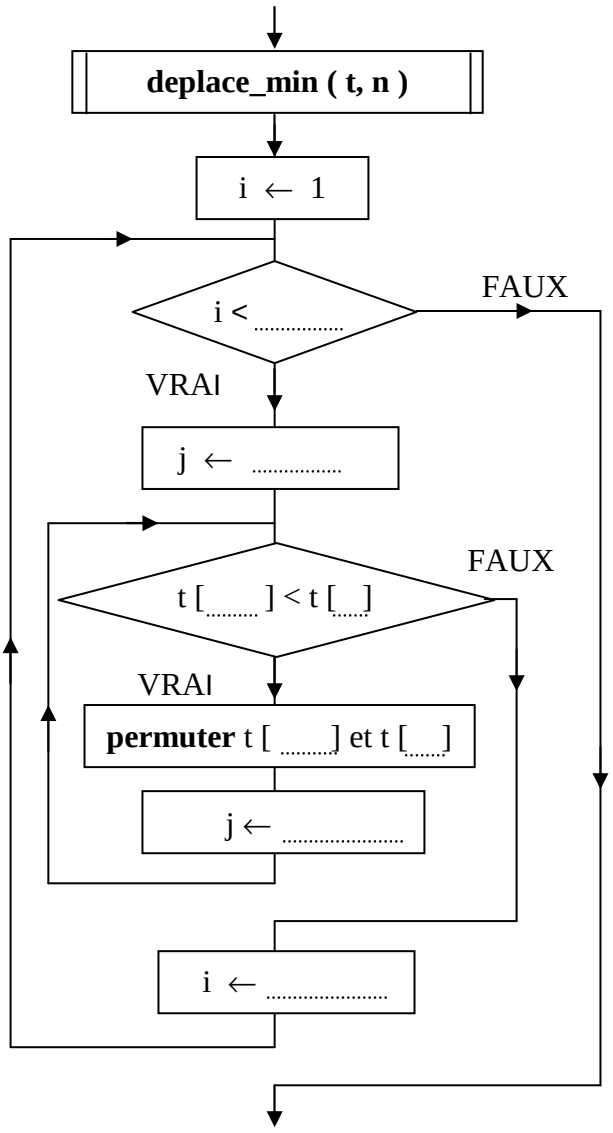
.....

.....

.....

.....

.....



- ✓ Développer la procédure **tri_bulle** dans le fichier **main_tri_num.c** et la tester en mode pas à pas avec les valeurs initiales de la variable principale **liste**.
- ✓ En utilisant le sous-programme **affiche_tableau** un résultat d'exécution pourra être :

```
Liste des valeurs non triees :

 1ere valeur:  5.00, 2eme valeur:  7.00, 3eme valeur:  8.00, 4eme valeur:  6.00,
 5eme valeur:  4.00,

Liste des valeurs triees par ordre croissant:

 1ere valeur:  4.00, 2eme valeur:  5.00, 3eme valeur:  6.00, 4eme valeur:  7.00,
 5eme valeur:  8.00,

Appuyer sur une touche pour continuer . . .
```

JEU PUISSANCE 4

1. Introduction

Puissance 4 est un jeu qui se joue à deux. Le but du jeu est d'aligner le premier 4 pions de sa propre couleur. Les 4 pions peuvent être alignés horizontalement, verticalement ou encore en diagonale.

2. Codage de la position des pions

Les pions sont déposés suivant **7 colonnes** qui contiennent **6 lignes**. Le **joueur 1** possède les pions noirs et le **joueur 2**, les pions blancs. Chacun des joueurs joue à tour de rôle. La figure ci-dessous montre un exemple de partie en cours ou le joueur 1 vient de gagner par alignement vertical de 4 pions noirs.

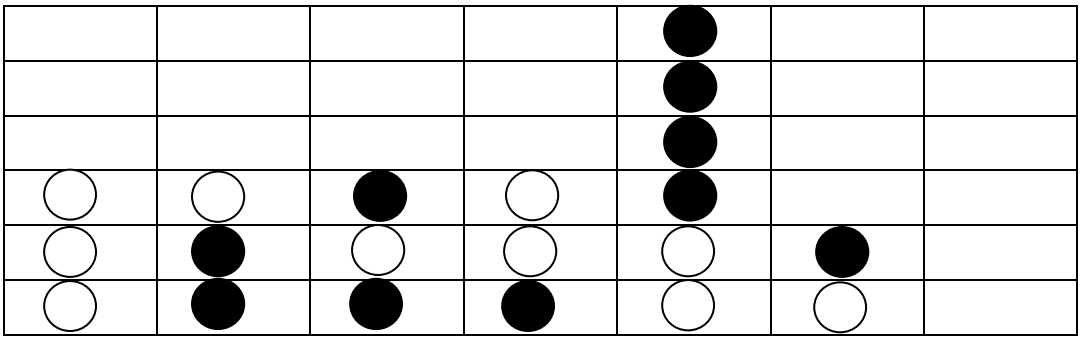


Figure 1 : **Partie en cours**

La grille de répartition des pions s'effectue à l'aide d'une variable tableau à 2 dimensions **Grille_jeu** [**i**][**j**] ou :

- ✓ **i** désigne le codage de la position verticale **i** = 5 : le pion est **au fond**
  i = 0 : le pion est sur la **première ligne**
- ✓ **j** désigne le codage de la position horizontale.

Le codage de la position des pions du **joueur 1** est effectuée en affectant la valeur **1** à **Grille_jeu** [**i**][**j**].

Le codage de la position des pions du **joueur 2** est effectuée en affectant la valeur **1** à **Grille_jeu** [**i**][**j**].

L'absence de pion se traduit par la valeur **0**

Ainsi, l'exemple de la partie présentée sur la **figure 1** est codée par la variable **Grille_jeu** de la manière suivante :

0	**0**	**0**	**0**	**0**	**1**	**0**	**0**
1	**0**	**0**	**0**	**0**	**1**	**0**	**0**
2	**0**	**0**	**0**	**0**	**1**	**0**	**0**
3	**-1**	**-1**	**1**	**-1**	**1**	**0**	**0**
4	**-1**	**1**	**-1**	**-1**	**-1**	**1**	**0**
5	**-1**	**1**	**1**	**1**	**-1**	**-1**	**0**
	0	**1**	**2**	**3**	**4**	**5**	**6**

Tableau 1

Un tableau à une dimension **Colonne_pleine [j]** **$0 \leq j \leq 6$** doit permettre de tester si une colonne est pleine de jetons. Cela permettra d'éviter d'essayer de remplir une colonne déjà pleine ! On prendra la convention suivante :

Si **Colonne_pleine [j] = 0** la colonne de rang **j** n'est pas pleine,

Si **Colonne_pleine [j] = 1** la colonne de rang **j** est pleine.

Dans l'exemple fournie dans la **figure 1**, le contenu de **Colonne_pleine** est :

j	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Colonne_pleine [j]	**0**	**0**	**0**	**0**	**1**	**0**	**0**

Cette variable permet donc de repérer que la colonne n°4 est pleine.

3. Structure du programme

3.1. Les données du programme principal

CONT

N_LIGN = 6

N_COLON = 7

TYPE

t_grille = Tableau [N_LIGN] [N_COLON]

VAR

Grille_jeu : t_grille

Colonne_pleine : Tableau [N_COLON] d'Entiers

Joueur : Entier /* Variable contenant 1 ou –1 pour différencier le tour des joueurs */

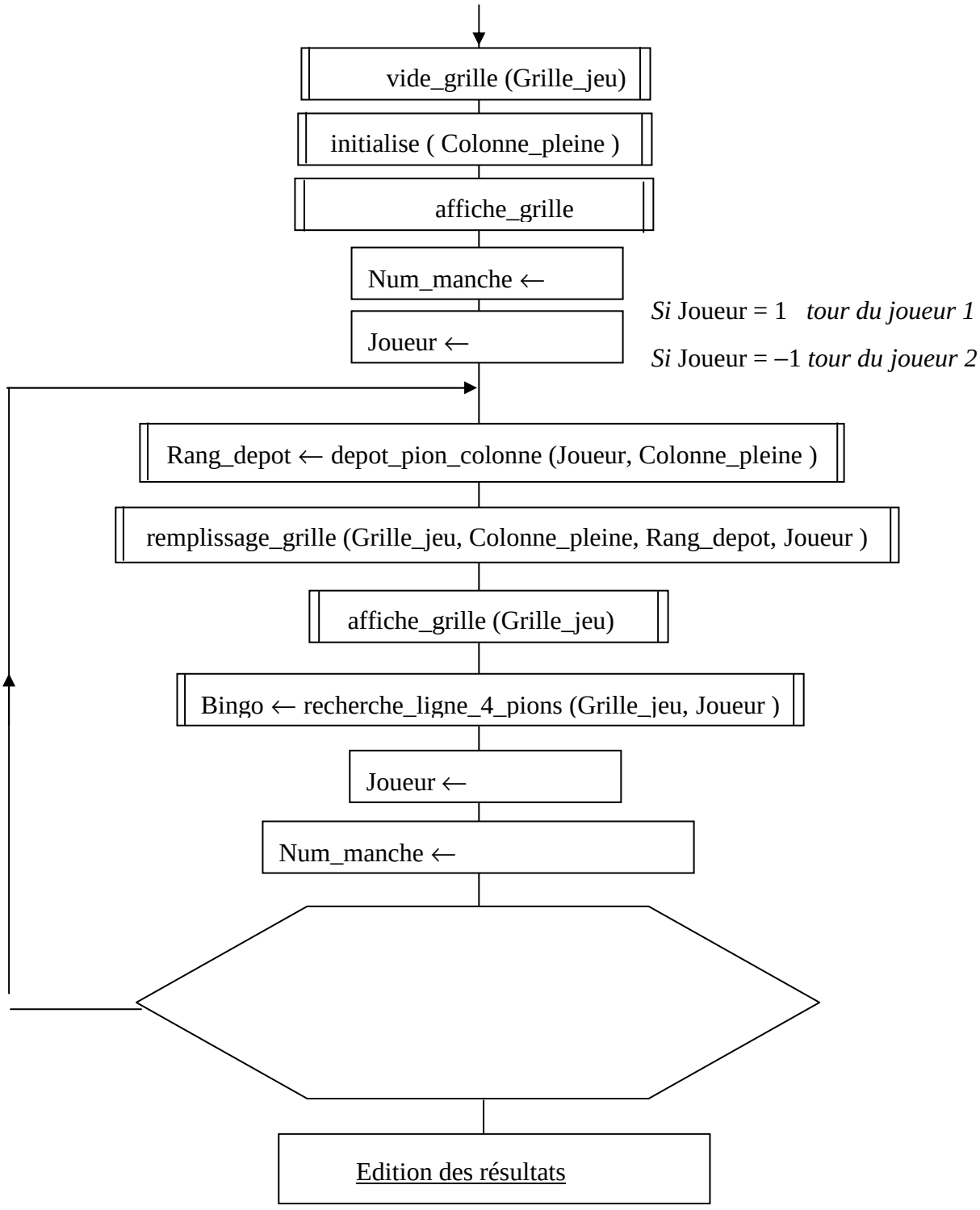
Rang_depot : Entier /*Numéro de colonne proposé par un joueur pour le dépôt de son jeton */

Bingo : Entier /* Indicateur permettant de détecter la réalisation d'une ligne de 4 pions consécutifs du même joueur : cette valeur vaut 4 lorsque la détection est effectuée avec la technique de recherche utilisée */

Num_manche : Entier /* Numéro de manche */

3.2. Le corps exécutable du programme principal

Il est demandé d'écrire les procédures **vide_grille**, **initialise**, **affiche_grille**, **remplissage_grille** ainsi que les fonctions **depot_pion_colonne** et **recherche_ligne_4_pions** qui seront utilisées suivant l'algorithme décrit par l'organigramme n°1.



Organigramme n°1

3.2.1. Procédure vide_grille

vide_grille (**Grille_jeu**)

Cette procédure doit permettre d'initialiser les valeurs de la variable Grille_jeu simulant une grille vidée de tout jeton.

3.2.2. Procédure initialise

initialise (**Colonne_pleine**)

Cette procédure doit permettre d'initialiser correctement les valeurs de la variable **Colonne_pleine**.

3.2.3. Procédure affiche grille

affiche_grille (**Grille_jeu**)

Cette procédure doit permettre d'afficher la grille avec des caractères permettant de voir l'évolution de la partie d'après le codage contenu dans la variable Grille_jeu tel qu'indiqué au paragraphe 4. On devra indiquer sur la ligne du bas les numéros de colonnes de la grille.
On aura intérêt à utiliser une variable locale **pion** de type **caractère** qui pourra être affecté des valeurs suivantes : ' ' : absence de pion,

'O' : présence d'un pion du Joueur n°1,

'X' : présence d'un pion du Joueur n°2.

3.2.4. Fonction depot_pion_colonne (fonction de saisie)

Cette fonction doit permettre de renvoyer dans la variable **Rang_depot**, la valeur du n° de colonne **saisie** par le joueur en cours.

Rang_depot ← **depot_pion_colonne** (**Joueur**, **Colonne_pleine**)

La valeur de la colonne saisie devra être sécurisée pour garantir que celle-ci est comprise entre **0** et **N_COLON**.
D'autre part, il faudra tenir compte de l'information apportée par la variable **Colonne_pleine** afin de proposer la saisie d'un autre n° de colonne dans le cas où une colonne serait déjà pleine.

3.2.5. Procédure remplissage_grille (Codage de l'ajout d'un jeton dans la grille)

remplissage_grille (**Grille_jeu**, **Colonne_pleine**, **Rang_depot**, **Joueur**)

Cette procédure doit permettre de placer le pion du joueur concerné à la colonne **Rang_depot**. Il suffit de rechercher la ligne qui n'est pas occupée par un pion afin de pouvoir le placer. La procédure devra également actualiser les valeurs de **Colonne_pleine** pour le cas où, après le dépôt du pion, une colonne se trouverait pleine.

3.2.6. Fonction recherche_ligne_4_pions

Bingo ← recherche_ligne_4_pions (**Grille_jeu**, **Joueur**)

Cette fonction doit permettre de détecter une ligne de 4 pions du Joueur 1 ou 2 en position verticale ou horizontale ou oblique. La valeur retournée est alors **4**.

- *Recherche d'une ligne verticale :*

La solution consiste à élaborer les sommes de 4 éléments verticaux contigus par colonne jusqu'à obtenir la valeur **4** (au maximum 3 somme par colonne)

- *Recherche d'une ligne horizontale :*

La solution consiste à élaborer les sommes de 4 éléments horizontaux contigus par ligne jusqu'à obtenir la valeur **4** (au maximum 4 sommes par ligne)

- *Recherche d'une ligne diagonale droite :*

La solution consiste à élaborer les sommes de 4 éléments diagonaux droits contigus jusqu'à obtenir la valeur **4** (au maximum 12 sommes)

- *Recherche d'une ligne diagonale gauche :*

La solution consiste à élaborer les sommes de 4 éléments diagonaux gauches contigus jusqu'à obtenir la valeur **4** (au maximum 12 sommes)

Remarque : la somme de chaque terme devra être pondérée par la valeur contenue dans le paramètre effectif **Joueur** (1 ou −1) afin d'obtenir toujours la valeur **4** en cas de détection.

Ainsi pour 4 valeurs –1 successives, on calcule :

$$(-1) \times (-1) + (-1) \times (-1) + (-1) \times (-1) + (-1) \times (-1) = 4$$

Pour 4 valeurs 1 successives, on calcule :

$$(1) \times (1) + (1) \times (1) + (1) \times (1) + (1) \times (1) = 4$$

3.2.7. Edition des résultats

En exploitant le contenu des variables **Bingo**, **Joueur** et **Num_manche**, Ecrire les instructions d'édition des résultats permettant d'indiquer si l'un des joueurs à gagné et en combien de manches où si il y a eu un match nul.

4. Résultat souhaité d'exécution lors du déroulement d'une partie

```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Joueur 1 : Depot du pion en colonne numero ? 0
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
| O |   |   |   |   |   |   |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Joueur 2 : Depot du pion en colonne numero ? 2
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
| O |   | X |   |   |   |   |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Joueur 1 : Depot du pion en colonne numero ? 10
Saisir un numero de colonne entre 0 et 6 : 2
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   | O |   |   |   |   |
| O |   | X |   |   |   |   |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

.
.
.
```

```
|   |   |   | O |   |   |   |   |
|   |   |   | X |   |   |   |   |
|   |   |   | O |   |   |   |   |
|   |   |   | X |   |   |   |   |
|   |   |   | O |   |   |   |   |
| O |   |   | X |   | X |   |   |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Joueur 1 : Depot du pion en colonne numero ? 2
Colonne pleine, choisissez une autre colonne : 5
|   |   |   | O |   |   |   |   |
|   |   |   | X |   |   |   |   |
|   |   |   | O |   |   |   |   |
|   |   |   | X |   |   |   |   |
|   |   |   | O |   |   |   |   |
| O |   |   | X |   | X | O |   |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
.
.
.
Joueur 2 : Depot du pion en colonne numero ? 6

|   |   |   | O |   |   |   |   |
|   |   |   | X |   |   |   |   |
|   |   |   | O |   |   |   |   |
|   |   |   | X | X |   | X |   |
|   |   |   | O | O |   | O | X |
| O |   |   | X | X | X | O | O |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Joueur 1 : Depot du pion en colonne numero ? 4

|   |   |   | O |   |   |   |   |
|   |   |   | X |   |   |   |   |
|   |   |   | O |   |   |   |   |
|   |   |   | X | X |   | X |   |
|   |   |   | O | O | O | O | X |
| O |   |   | X | X | X | O | O |
-----------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Le joueur 1 a gagné en 17 manches

Appuyer sur une touche pour continuer ...
```

ENVIRONNEMENT DE DEVELOPPEMENT INTEGRE DEV-C++

1. Introduction

L'environnement de développement intégré Dev-C++4.9.9.1 (Bloodshed Software) téléchargeable gratuitement sur le site **http://www.bloodshed.net/** intègre plusieurs utilitaires :

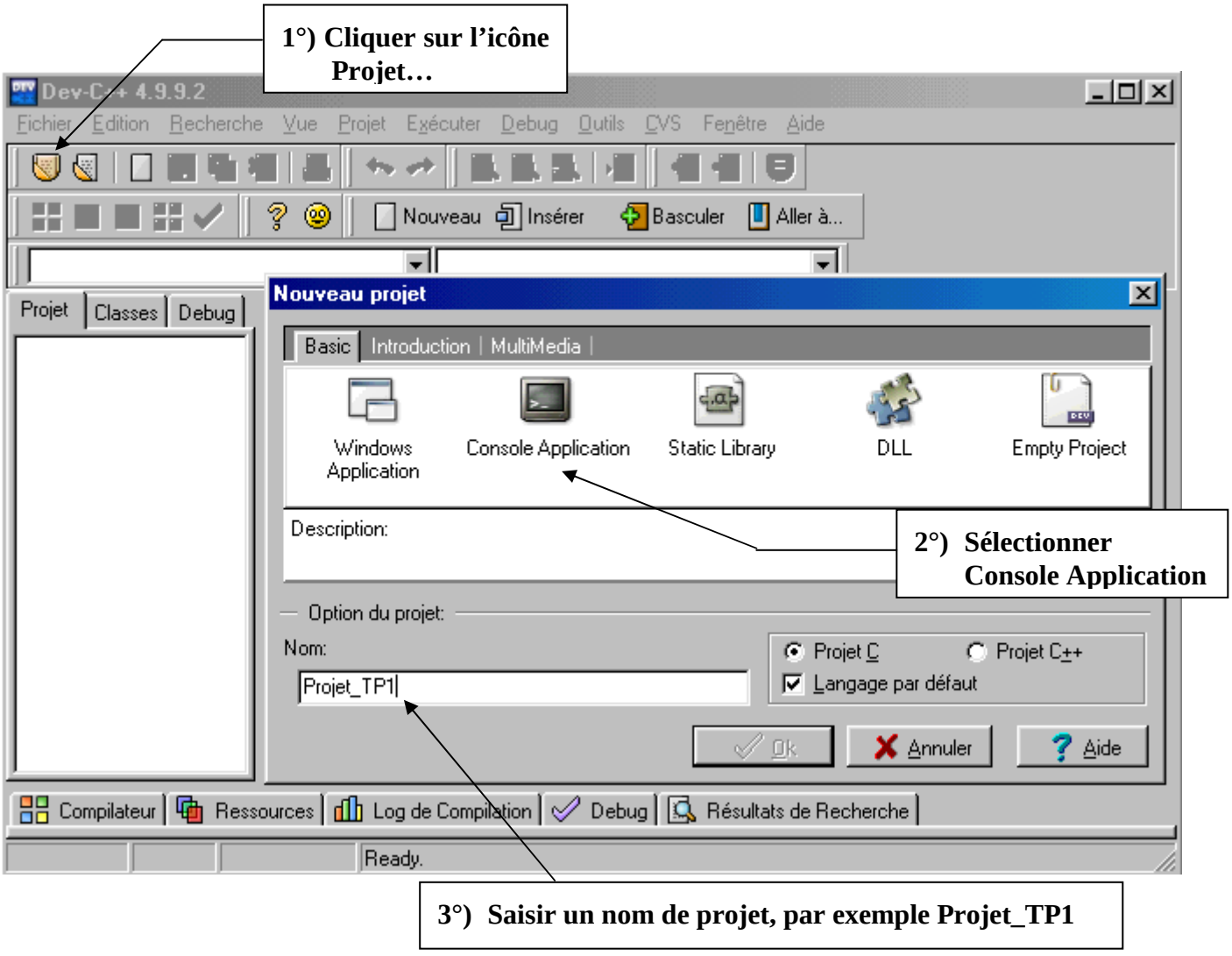
- un éditeur de texte permettant la saisie des textes sources en C (création de fichiers **`*.c`**),
- un compilateur et un éditeur de liens permettant de traduire le texte source en programme exécutable (fichier **`*.exe`**),
- un débogueur permettant d'exécuter un programme pas à pas et d'examiner l'évolution du contenu des variables. C'est un moyen précieux pour trouver les erreurs algorithmiques.

2. Lancement de Dev-C++ à partir de Windows :

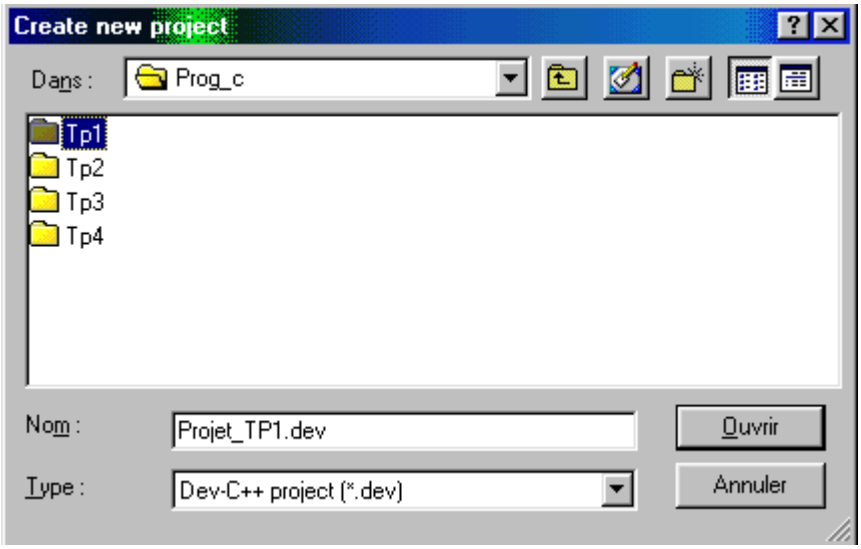


Cliquer sur l'icône correspondant.

On obtient la fenêtre suivante :

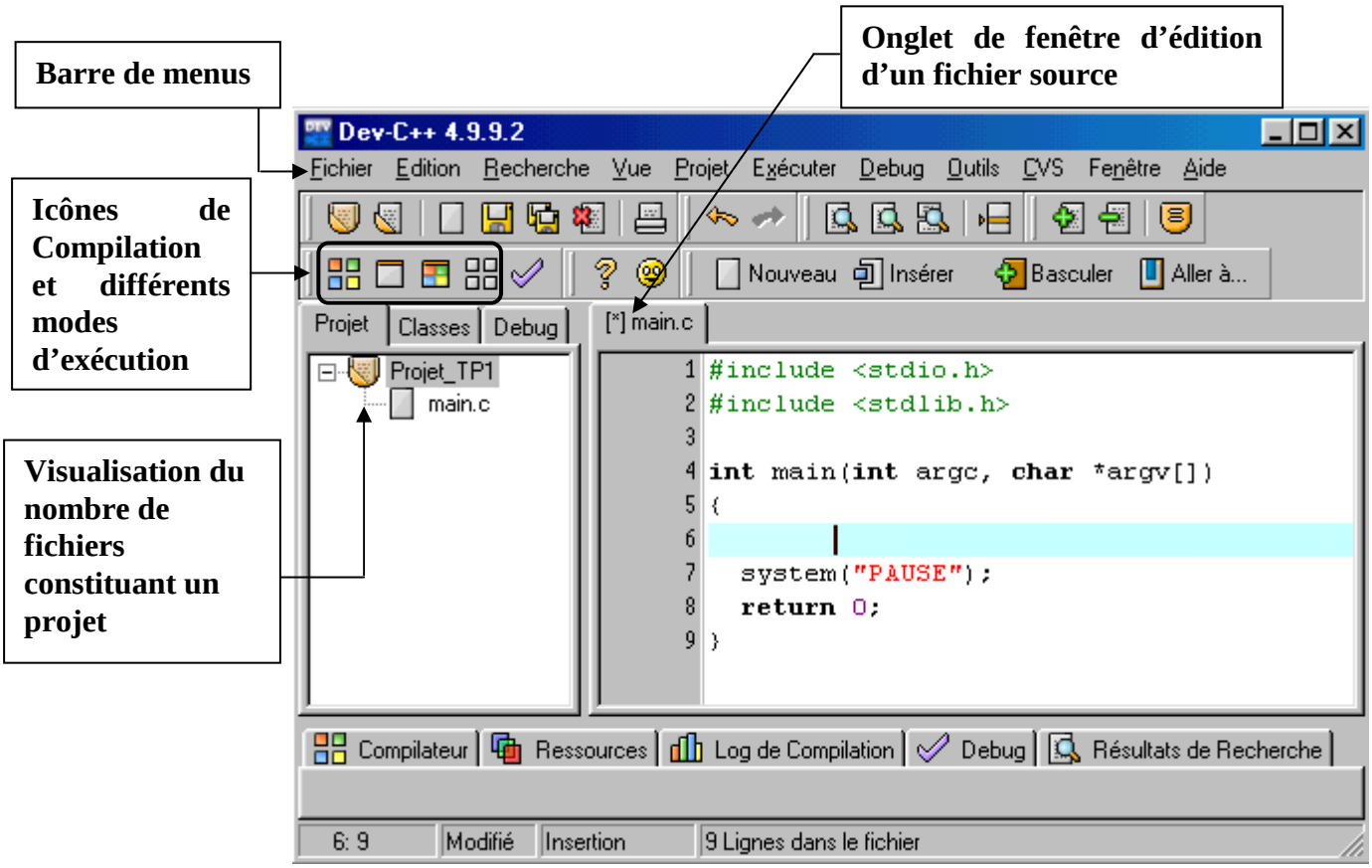


Une fois le nom du projet saisi, on sauvegarde le nom du fichier projet (d'extension **`.dev`**) dans un des répertoires désirés :



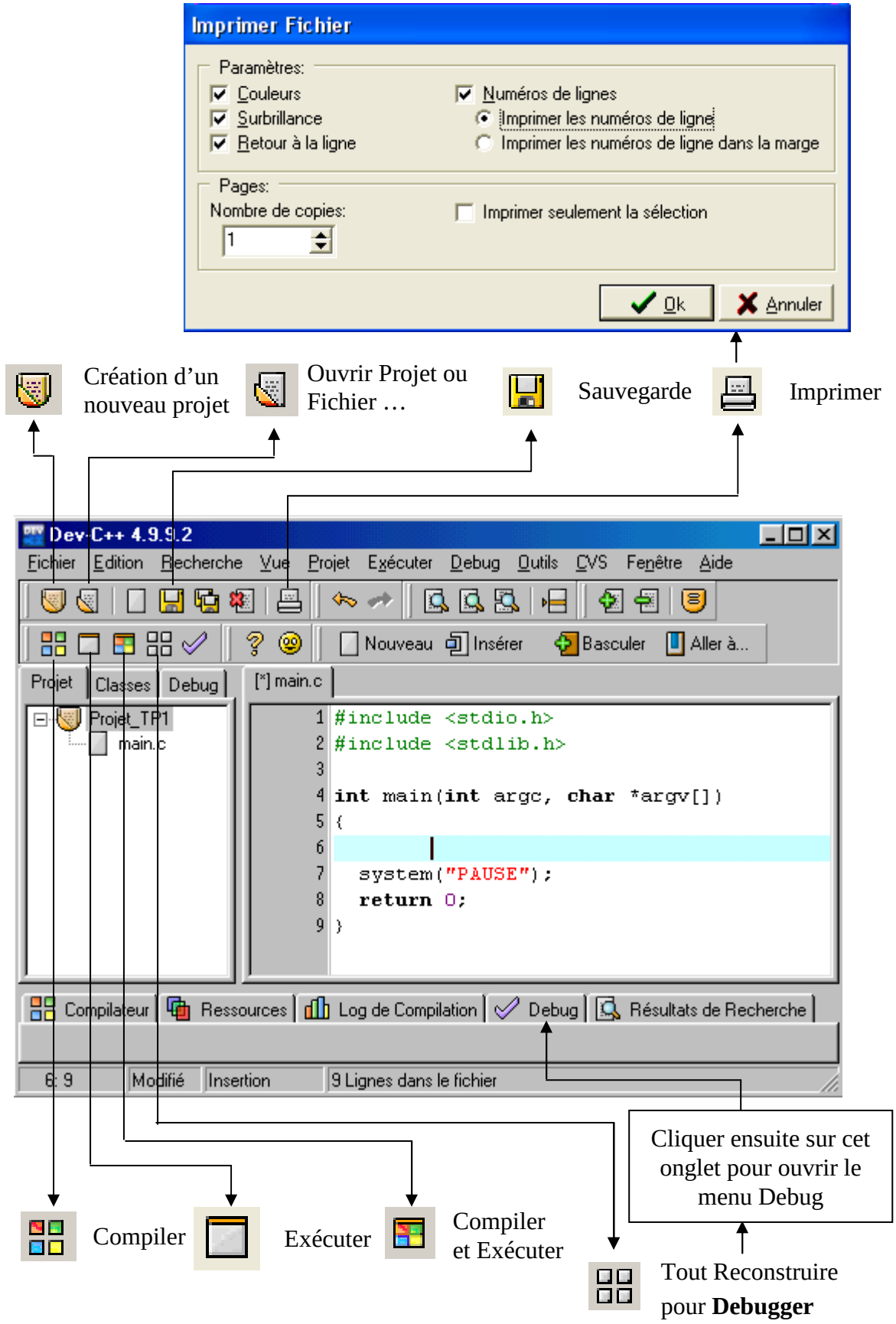
On obtient alors systématiquement un fichier baptisé **main.c** par défaut qui contient le canevas du programme principal, contenant la fonction principale **`main`**. On aura intérêt à le rebaptiser (par exemple : **main_projet_tp1.c**)

3. Présentation de l'environnement Dev-C++

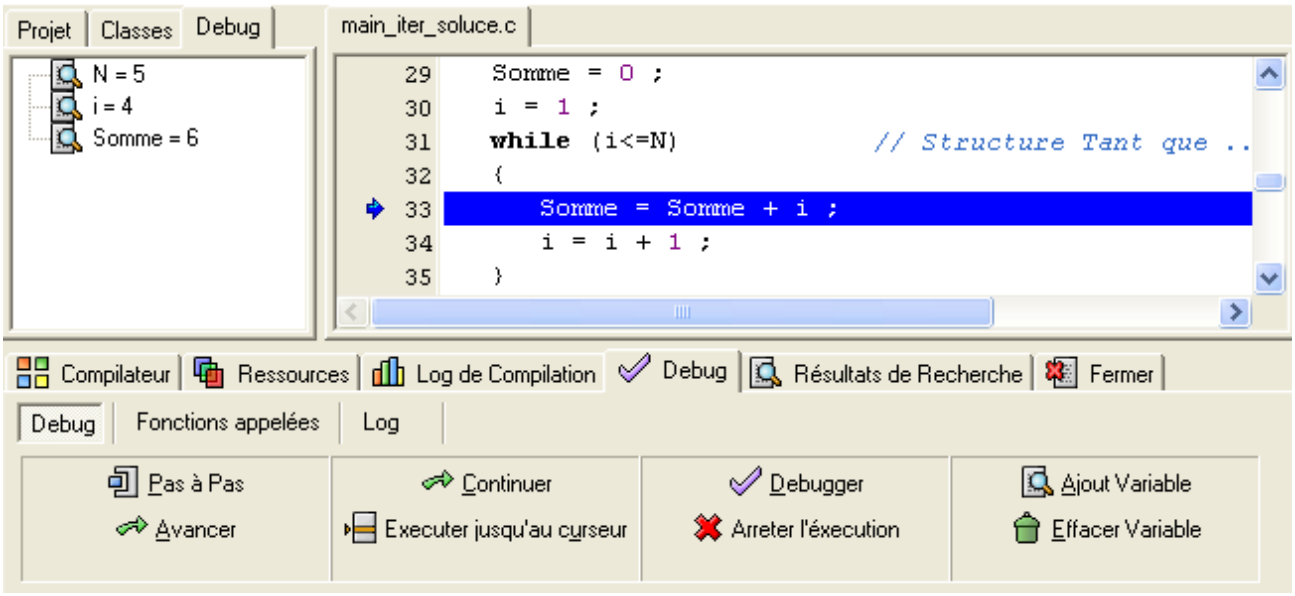


Dans la suite, on ne détaillera pas toutes les fonctionnalités, les menus étant suffisamment explicites.

3.1. Les differentes icônes directement accessibles



3.2. Les commandes d'exécution en mode pas à pas (menu debug)



(F8)

Pas à Pas

Exécution de l'instruction suivante

Continuer

Finir le programme (abandon du mode pas à pas)

Ajout Variable

Ajout de variable

(F7)

Avancer

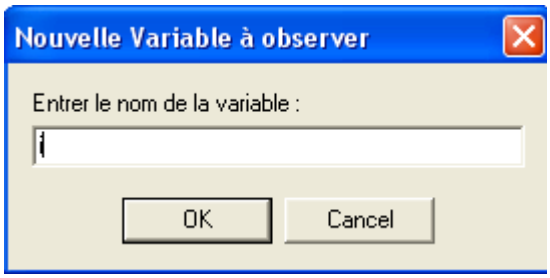
Entrée dans une fonction ou procédure

Executer jusqu'au curseur

Exécution jusqu'à l'instruction placée sur le curseur.

Arreter l'éxecution

Arrêt de l'exécution



3.3. Résumé des actions de compilation

