Introduction de la programmation   
Projet Premier semestre  
Sujet : Guide des prénoms de Bordeaux

**Justifications techniques**

Contenu

[A. Fonction principale 3](#_Toc409018358)

[a. La structure Prenom (cf. annexe 1) 3](#_Toc409018359)

[b. Le menu (cf. annexe 2) 3](#_Toc409018360)

[B. Affichage des textes 3](#_Toc409018361)

[a. La structure Texte (cf. annexe 3) 3](#_Toc409018362)

[b. Gestion de l’affichage 4](#_Toc409018363)

[c. Choix de la langue 4](#_Toc409018364)

[C. Choix du prénom 5](#_Toc409018365)

[a. Tri du tableau 5](#_Toc409018366)

[b. La recherche par dichotomie 5](#_Toc409018367)

[ANNEXE 6](#_Toc409018368)

[Annexe 1 : La structure prénom : 6](#_Toc409018369)

[Annexe 2 : Code de la boucle du Menu 7](#_Toc409018370)

[Annexe 3 : La structure Texte 8](#_Toc409018371)

[Annexe 4 : La fonction affichageTexte 8](#_Toc409018372)

[Annexe 5 : Le tri rapide 9](#_Toc409018373)

# Fonction principale

La fonction principale dans notre programme est la fonction **main**. Cette dernière permettra de charger l’ensemble des données contenues dans le fichier texte, de les mettre sous forme de structure et de les stockées dans un tableau

## La structure Prenom (cf. annexe 1)

Lors de l’initialisation de notre programme, chaque ligne du fichier texte est traitée et stockée dans une structure **Prenom** qui est composé de 4 variables :

* Un entier **annee** qui contiendra l’année du prénom
* Une chaîne de caractère **prenom**
* Un entier **nombrePrenom** qui contiendra les nombre de nouveau-né portant ce prénom sur une année.
* Un entier **ordre** qui contiendra l’ordre sur 100 de ce prénom

Nous avons choisi ce format de structure car c’est celui qui se rapproche le plus de la présentation es informations d’origines. Nous stockons ensuite l’ensemble de ces structures dans un fichier texte.

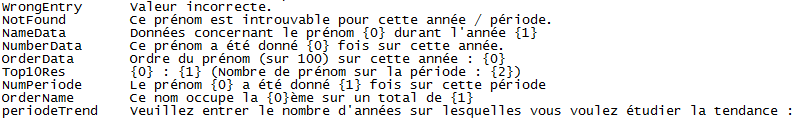
## Le menu (cf. annexe 2)

Une fois les traitements d’initialiser du programme terminé nous avons mis en place un menu. Ce menu prend la forme **d’une boucle while** qui tourne en boucle jusqu’à ce que l’utilisateur quitte le programme.  
Dans ce menu nous proposons les différents choix possible à l’utilisateur puis en fonction de sa réponse, nous utilisons une **structure switch** afin de lancer le traitement voulue. Une fois le traitement terminé nous réaffichons ce menu.

# Affichage des textes

## La structure Texte (cf. annexe 3)

Afin d’optimiser au maximum notre programme nous avons décidé d’utiliser un système particulier pour gérer l’affichage des informations qui apparaîtront à l’écran. Nous avons stockés l’ensemble des textes de notre programme dans un fichier texte qui possèdent la structure suivante :



Le premier mot correspond à **l’identifiant** du texte, puis après une tabulation, on trouve le texte qui sera affiché à l’écran.  
Lors de l’initialisation de notre programme, nous lisons ce fichier texte et stockons chaque ligne dans une structure **Texte**  qui possède les variables suivantes :

* Une chaîne de caractères **nomTexte** qui contiendra l’identifiant du texte
* Une chaîne de caractères **texte** qui contiendra le texte à afficher

Nous stockons ensuite ces structures dans un tableau de **Texte.**

## Gestion de l’affichage

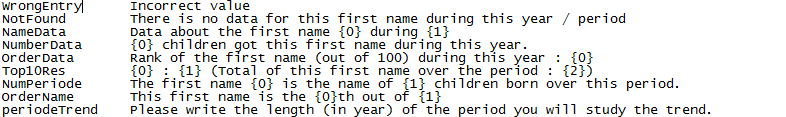
Afin de gérer l’affichage grâce à nos structures **Texte**, nous avons eu besoin de créer une nouvelle fonction. C’est le rôle de la fonction **affichageTexte(cf. annexe 4)**, cette fonction prendra en paramètre l’identifiant du texte que l’on souhaite afficher, et après une recherche dans le tableau contenant l’ensemble de nos structure **Texte**, affichera le texte correspondant à l’identifiant.

En utilisant le mot clé **params** dans la définition de la fonction, il nous a été possible de créer un tableau variable qui contiendra l’ensemble des paramètres que l’on souhaite afficher dans la chaîne de caractère. Par défaut, si l’identifiant n’est pas trouvé dans le tableau, la fonction affichera ce dernier.

L’avantage de cette méthode est qu’on désolidarise en partie le texte affiché du programme. En effet si l’on souhaite modifier le texte il suffit de modifier directement le fichier texte et non plus le texte en brut dans le programme.

## Choix de la langue

L’autre intérêt de cette méthode est la mise en place et le choix de la langue. En effet, dans le cadre de notre programme par exemple nous avons rajoutés l’anglais comme langue disponible, cela s’est traduit par l’ajour d’un second fichier langues qui possède les même identifiant pour le début de chaque ligne mais dont le texte affichés sera différents.



Ainsi si l’on souhaite traduire notre programme dans une autre langue, au lieu de traduire en brut dans le programme ou de recopier une traduction faite par un tiers, le développeur aura juste à rajouter une option correspondant à la nouvelle langue et donc au nouveau fichier.

Actuellement le souci majeur de notre solution est que les fichiers contenant le texte sont non cryptés et donc modifiable. Pour sécuriser un minimum notre programme et donc cette solution, il faudrait crypter ces fichiers de langues.

# Choix du prénom

Afin de donner à l’utilisateurs la possibilité ou non de choisir un prénoms lors de l’utilisation de l’application, nous lui laissons la possibilité d’entrer lui-même un prénom ou de laisser le programme choisir un prénom de façon aléatoire. Afin d’optimiser au maximum la recherche des informations lorsque l’utilisateur rentre lui-même un prénom, il nous a fallu intégrer une fonction de tri et de rechercher sur un tableau.

## Tri du tableau

Au vu du grand nombre de données que nous devions traiter, il nous fallut chercher un algorithme performant afin de permettre le tri du tableau. Nous nous sommes donc pencher sur la complexité algorithmique des différents algorithme de tri existant. Nous avons dans un premier temps effectué un algorithme de tri à bulle simple à mettre en place mais avec une complexité élevée : en moyenne. De plus le temps d’exécution du était relativement long, de l’ordre de 15 à 20 secondes sur l’ensemble des données.

Nous nous sommes tournés vers un algorithme plus rapide, le tri rapide (quick sort). Cette méthode de tri consiste à mettre un élément du tableau à sa place définitive (le pivot), en permutant les éléments de façon de telle sorte que tous les éléments inférieurs au pivot soient à gauche et que tous ceux qui sont supérieurs à droite. Cette partie s’appelle le partitionnement

On va ensuite redéfinir un pivot pour chacune des parties créées (à gauche et à droite du pivot) puis on va appliquer une nouvelle fois le partitionnement sur ces deux sous tableaux.  
Ce processus est alors répété récursivement jusqu’à ce que le tableau soit complétement trié.

La complexité d’un tel algorithme est de en moyenne et de complexité quadratique ( dans le pire des cas. Cet algorithme est de plus beaucoup plus rapide et tri l’ensemble des données sans temps de traitement visible lors de nos tests.

Le code de l’algorithme est présent en annexe 5

## La recherche par dichotomie

Afin d’optimiser également le temps de recherche sur ce tableau trié, nous avons mis en place un algorithme de recherche dit par dichotomie.  
Cet algorithme va chercher la case situé au milieu du tableau puis va la comparer à la valeur recherchée. Si la valeur de la case est inférieure à la valeur recherché on répétera la même opération mais dans la moitié supérieur à cette case. Dans le cas inverse on refera la même opération dans la moitié inférieure.  
On va continuer de répété cette opération jusqu’à ce que l’on trouve la valeur chercher ou que l’on est exploré toute les possibilités.

Nous avons choisi d’utiliser cette algorithme de recherche premièrement car nous travaillions sur un tableau que nous avons trié. Ensuite cette algorithme est très performant en terme de complexité, en effet, la complexité de cette algorithme sera de dans le pire des cas ce qui est relativement faible.

# ANNEXE

## Annexe 1 : La structure prénom :

public struct Prenom

{

public string prenom;

public int annee;

public int nombrePrenom;

public int ordre;

};

## Annexe 2 : Code de la boucle du Menu

while (!quitte)

{

Console.Clear();

Console.ForegroundColor = ConsoleColor.Green;

affichageTexte("TitleLine1", texteProgramme);

affichageTexte("TitleLine2", texteProgramme);

affichageTexte("TitleLine3", texteProgramme);

Console.ResetColor();

affichageTexte("MenuLine1", texteProgramme);

affichageTexte("MenuChoice1", texteProgramme);

affichageTexte("MenuChoice2", texteProgramme);

affichageTexte("MenuChoice3", texteProgramme);

affichageTexte("MenuChoice4", texteProgramme);

affichageTexte("MenuChoice5", texteProgramme);

affichageTexte("MenuChoice0", texteProgramme);

choixOk = false;

while (!choixOk){

choixOk = true;

try{

affichageTexte("EnterChoice", texteProgramme);

choix = int.Parse(Console.ReadLine());

}

catch{

affichageTexte("IncorrectValue", texteProgramme);

choixOk = false;

}

}

switch (choix){

case 1:

prenomSurUneAnnee(Donnees, texteProgramme);

break;

case 2:

top10PrenomsPeriode(Donnees, texteProgramme);

break;

case 3:

nomPeriodeDonnee(Donnees, texteProgramme);

break;

case 4:

tendancePrenom(Donnees, texteProgramme);

break;

case 5:

langue = choixLangue();

recuperationTexteProgramme(langue, texteProgramme);

break;

case 6:

triRapideSurPrenom(DonneesTrieSurPrenom, 0, DonneesTrieSurPrenom.Length - 1);

for (int k = 0; k < 500; k++){

Console.WriteLine(DonneesTrieSurPrenom[k].prenom);

}

break;

case 0:

quitte = true;

break;

}

## Annexe 3 : La structure Texte

public struct Texte

{

public String nomTexte;

public String texte;

}

## Annexe 4 : La fonction affichageTexte

public static void affichageTexte(String texte, Texte[] texteProgramme, params String[] valeur)

{

bool trouve = false;

int i = 0;

while (trouve == false && i < texteProgramme.Length)

{

if (texte == texteProgramme[i].nomTexte)

{

trouve = true;

switch (valeur.Length)

{

case 1:

Console.WriteLine(texteProgramme[i].texte, valeur[0]);

break;

case 2:

Console.WriteLine(texteProgramme[i].texte, valeur[0], valeur[1]);

break;

case 3:

Console.WriteLine(texteProgramme[i].texte, valeur[0], valeur[1], valeur[2]);

break;

case 4:

Console.WriteLine(texteProgramme[i].texte, valeur[0], valeur[1], valeur[2], valeur[3]);

break;

default:

Console.WriteLine(texteProgramme[i].texte);

break;

}

}

i++;

}

if (trouve == false)

Console.WriteLine(texte);

}

## Annexe 5 : Le tri rapide

Fonction de partitionnement :

public static int partitionTriRapide(Prenom[] Donnees, int premier, int dernier, int pivot)

{

Prenom temp;

temp = Donnees[dernier];

Donnees[dernier] = Donnees[pivot];

Donnees[pivot] = temp;

int j = premier;

for (int i = premier; i <= dernier - 1; i++)

{

if (String.Compare(Donnees[i].prenom, Donnees[dernier].prenom) < 0)

{

temp = Donnees[i];

Donnees[i] = Donnees[j];

Donnees[j] = temp;

j++;

}

}

temp = Donnees[dernier];

Donnees[dernier] = Donnees[j];

Donnees[j] = temp;

return j;

}

Fonction de tri :

public static void triRapideSurPrenom(Prenom[] Donnees, int premier, int dernier)

{

int pivot;

if (premier <= dernier)

{

pivot = premier;

pivot = partitionTriRapide(Donnees, premier, dernier, pivot);

triRapideSurPrenom(Donnees, premier, pivot - 1);

triRapideSurPrenom(Donnees, pivot + 1, dernier);

}

}