UNIVERSITÉ FRANÇOIS - RABELAIS
CAMPUS BLOIS

UFR Sciences et Techniques

IUP Blois
Licence S&T – 1°année

UE 102 – Informatique

Partie I — Architecture des ordinateurs

TRAVAUX DIRIGES MACHINE

Enseignant **Jean-Yves ANTOINE**
(Jean-Yves.Antoine AT univ-tours.fr)

TP 1 — Découverte de l'environnement de programmation Turbo Pascal 7

J.Y. Antoine

Tout au long de ce semestre, vous allez travailler sur l'environnement Turbo Pascal 7.0 pour DOS de Borland, utilisable sous Windows. Prévu initialement pour le système d'exploitation DOS (d'où le côté très fruste de son interface), il s'agit d'un environnement intégré, c'est à dire que l'écriture de vos programmes et leur exécution sera effectué à partir du même logiciel. Tout au long de vos activités de développement, vous saisirez donc votre programme à l'aide de l'éditeur de texte intégré à l'environnement, de même que vous pourrez compiler et exécuter ces programmes à partir de l'environnement.

Si cet environnement de programmation est très fruste, il a l'avantage de coller le plus fidèlement à norme ANSI de langage Pascal, et est surtout totalement libre de droit. Vous pouvez donc télécharger à des fins personnelles cet environnement, à partir du site WWW suivant :

http://www.simonhuggins.com/courses/progbegin/pascal/download/#download

D'autres versions antérieures de Turbo Pascal peuvent être trouvées à l'adresse : http://community.borland.com/article.

Tout au long des TPs, votre activité consistera à éditer des programmes et les compiler de manière alternée : en cas d'erreur, vous devez retourner sous l'éditeur pour modifier votre programme pour ensuite le compiler et le tester.

Nous allons décrire les différentes opérations qui peuvent résumer une session type en se basant pour commencer sur l'exemple de convertisseur que nous avons étudié en cour.

1. Convertisseur Euro / Franc

Votre première tâche va consister à saisir le programme que nous avons vu en fin de cours. Pour cela, il vous faut lancer l'environnement Turbo Pascal 7.0 à partir du menu démarrer. Une « superbe » fenêtre DOS apparaît. La barre de menu supérieure vous permettra d'éditer, sauvegarder, compiler et exécuter vos programmes. La zone de travail en dessous correspond à la zone de texte dans laquelle vous allez écrire (et modifier) votre programme.

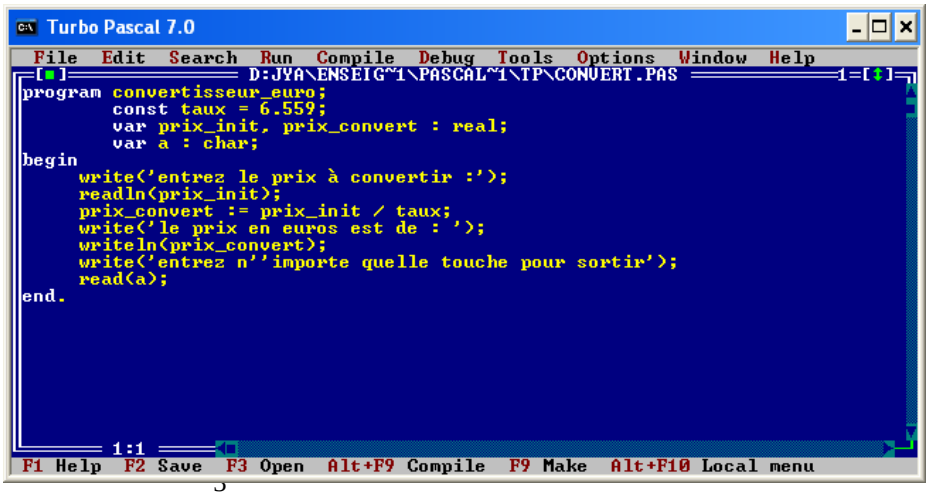
1.1. Ecriture du programme

1.1.1. Nouveau programme — Dans un premier temps, vous allez rédiger un nouveau programme. Il vous donc ouvrir un nouveau fichier où celui-ci sera enregistré : dans le menu File, choisissez-donc l'option New. La zone de travail change de couleur et s'appelle NONAME.PAS : c'est pour le moment le nom par défaut de votre fichier. Notez que par défaut, les programmes Pascal ont pour extension .PAS.

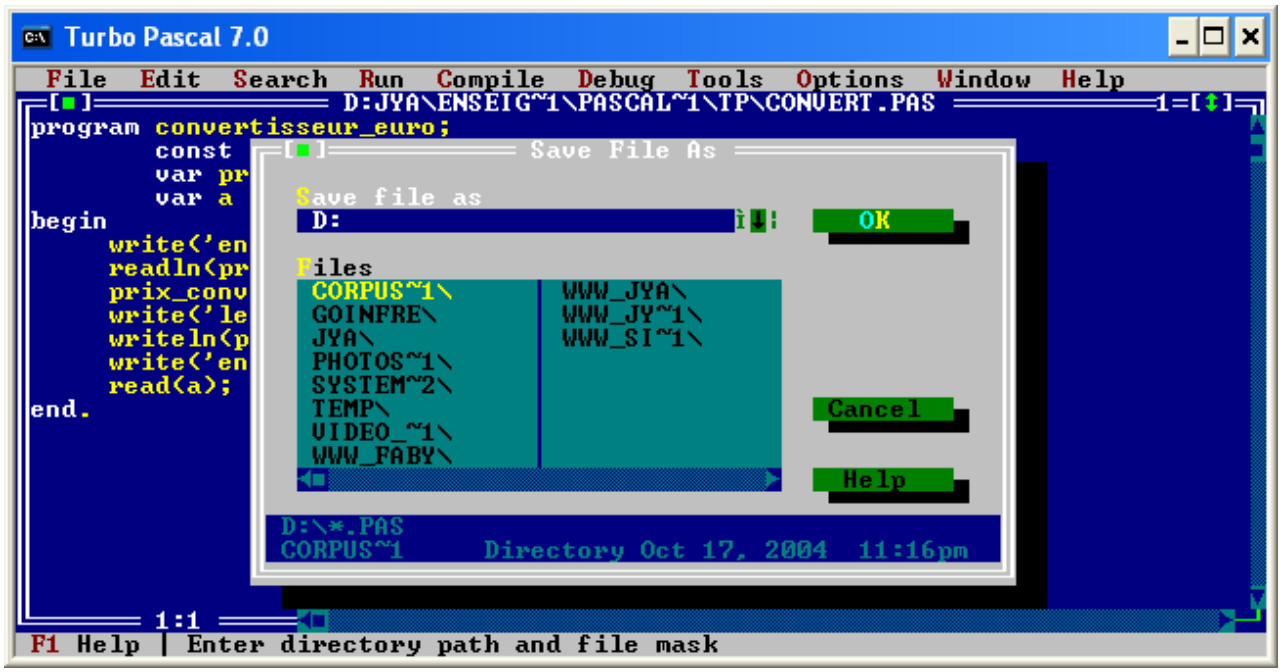
1.1.2. Ecriture du programme — Ecrivez alors le programme vu en cours dans cette zone de travail, à la manière de tout éditeur de texte. Il est recommandé de structurer aussi proprement que possible les différentes parties de votre code (utilisez les décalages pour aligner les éléments de même niveau) afin de le rendre aussi lisible que possible.

Au fil de la saisie, vous remarquez que l'analyseur lexical de Turbo Pascal reconnaît les mots réservés du langage (`var`, `begin`,…) : ceux-ci apparaissent automatiquement dans une autre couleur, ce qui rend le programme d'autant plus lisible.

Au final, votre programme ressemble à un texte équivalent à ce que l'on voit sur l'image à droite. Vous allez donc pouvoir l'enregistrer (on n'est jamais trop prudent…) avant de tenter de la compiler et l'utiliser.



1.1.3. Enregistrement du programme — Pour le moment, votre programme a un nom peu explicite (NONAME). Nous allons donc l'enregistrer dans un fichier portant un nom plus parlant. Pour cela, sélectionnez dans le menu `File` l'option `Save As` . Une boite de dialogue apparaît, qui vous demande le nom de votre fichier. Sélectionnez tout d'abord la partition sous laquelle vous allez sauvegarder votre fichier (par exemple D : ou F:) dans la ligne `Save File As` :



Puis naviguez dans l'explorateur de fichier sommaire (zone de sélection `Files`) pour atteindre le répertoire de votre choix. **Attention** : vous êtes en présence d'un environnement conçu pour le système d'exploitation DOS, pour qui les noms de répertoire ou de fichiers sont limités à 8 caractères : un répertoire Windows SYSTEMES_ENSEIGNEMENT apparaîtra donc tronqué sous la forme : SYSTEM~1 par exemple.

Une fois que vous êtes positionnés dans le bon répertoire, vous pouvez écrire le nom de votre fichier dans la zone `Save file As`. Ici encore, pensez à donner à votre fichier un nom de 8 caractères maximum, et lui ajouter une extension « PAS ». Par exemple, dans notre exemple : `CONVERT.PAS`

Cliquez sur OK : votre programme est enregistré. A l'avenir, lorsque vous modifierez ce fichier programme, il vous suffira de sélectionner l'option `Save` du menu `File` pour mettre à jour votre fichier (ancienne version écrasée).

1.2. Compilateur

Notre programme étant désormais rédigé, nous allons le traduire en langage machine. Pour cela, nous allons le compiler à l'aide de l'option Compile du menu … Compile. Une boite de dialogue apparaît qui vous précise que la compilation s'est bien déroulée et vous donne la taille du code exécutable correspondant. Si tel est le cas, vous allez pouvoir exécuter ce code.

Sinon, votre programme comporte des erreurs de syntaxe : la boite de dialogue vous donne des indications sur les sources d'erreurs prévisibles, avec la ligne où celles-ci ont été détectées. **Attention**, le compilateur essaie de comprendre ce que vous avez voulu dire, ses conseils sont parfois peu judicieux). A vous de les comprendre, de modifier votre programme et le recompiler jusqu'à ce que votre code soit correct.

1.3. Exécution

Si votre programme passe l'étape de la compilation, c'est que sa **syntaxe** est correcte. C'est à dire qu'il respecte la grammaire du langage Pascal. Cela ne veut pas dire pour autant que ce programme a une **sémantique** correcte, c'est-à-dire qu'il fait ce qu'on attend de lui. Seules de multiples tests (exécution de jeux d'essais) vont nous en assurer.

Pour cela, lancer l'exécution du programme à l'aide de l'option `Run` du menu `Run`. Une fenêtre de dialogue apparaît, dans laquelle vous pourrez suivre l'exécution du programme. Afin de vous assurer que le programme fonctionne bien, testez celui-ci aussi exhaustivement que possible, en essayant d'imaginer toutes les situations possibles :

- Fonctionnement normal : on rentre un prix, soit entier, soit réel, et le programme fait la conversion,
- Fonctionnement en situation atypique : l'utilisateur rentre un nombre négatif, un caractère etc…

Pour cela, vous lancerez donc autant de session d'exécution que nécessaire.

Il est très important de tester un programme dans des conditions atypiques. En effet, le concepteur d'un programme a pour habitude de ne prévoir que les cas d'utilisation favorables pour lesquels le logiciel est prévu. Cependant, il est fréquent que l'utilisateur fasse une erreur, et il est essentiel que le logiciel gère parfaitement ce genre de situation. En particulier, une erreur de l'utilisateur ne doit pas avoir de conséquences dommageables, de même que le logiciel doit pouvoir aider l'utilisateur à corriger son erreur.

Dans le cas de notre petit programme, l'enjeu des situations atypiques est minime. Mais autant prendre dès le départ de bonnes pratiques et penser à intégrer dès le départ la recommandation « *design for errors* » dans sa programmation. A l'avenir, vous devrez donc systématiquement :

- prévoir dans votre programme la gestion des erreurs et autres situations atypiques
- tester exhaustivement votre programme dans toutes les situations.

2. Compilation : correction d'erreurs

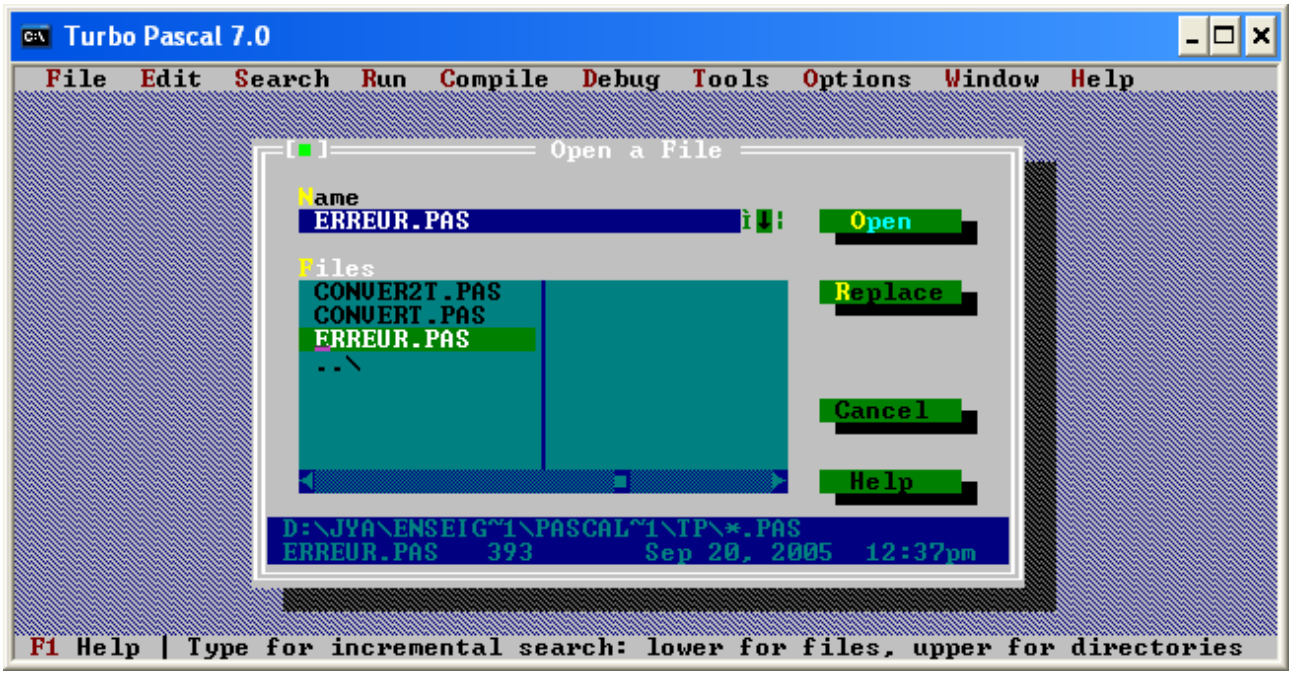
Si vous avez correctement saisi votre programme de conversion en euros, vous n'avez rencontré aucun message d'erreurs lors de l'étape de compilation. En guise de découverte de l'environnement Turbo Pascal, il n'est peut-être pas inutile de se confronter à ces situations problématiques. Pour cela, nous allons travailler sur un programme de conversion déjà saisi dans un fichier et qui comporte à dessein de nombreuses erreurs de syntaxe.

2.1. Chargement d'un programme existant

Le programme en question est enregistré dans le fichier ERREUR.PAS, présent sur le site WWW de ce cours :

`http://www.info.univ-tours.fr/~antoine/Pascal.html`

Récupérez ce fichier et enregistrez le sous un répertoire de votre choix. Chargez ensuite ce programme sous l'environnement Turbo Pascal à l'aide de l'option `open` du menu `File`. Une fenêtre de dialogue analogue à celle utilisée pour la sauvegarde de fichier apparaît à l'écran.



Pour accéder au bon répertoire, il vous faudra peut-être ici également préciser la partition sous laquelle se trouve votre répertoire. Cette information se précise dans le champ `Name` de la fenêtre de dialogue. Une fois cette information entrée, vous naviguez dans l'explorateur de fichier pour trouver le fichier recherché. Après avoir cliqué le bouton Open, le programme correspondant s'affiche dans la zone de travail. A partir de là, vous pouvez modifier, compiler et exécuter le programme comme précédemment.

2.2. Correction du programme

Compilez ce fichier et identifiez, à partir des messages d'erreur fournis, la cause des différentes erreurs de syntaxe qui ont été insérées intentionnellement. Modifiez le programme jusqu'à ce que celui-ci soit enfin exécutable.

3. A vous de jouer : mes premiers programmes Pascal

Vous savez désormais utiliser l'environnement de programmation Turbo Pascal. Il vous est donc demandé de réaliser ex-nihilo les programmes ci-dessous, qui feront également appel à vos connaissances en géométrie.

1. Ecrire un programme CERCLE.PAS qui, une fois saisi au clavier la longueur du rayon d'un cercle, donne son périmètre et sa surface.
2. Ecrire un programme CHEVRE.PAS qui calcule la surface accessible par une chèvre attachée au grand côté d'une maison de dimension L x l (valeurs que l'on saisira au clavier), par une corde le longueur L + l.

4. Pour aller plus loin : branchement conditionnel et itération

Cet exercice fait appel à la notion de branchement conditionnel et de boucle d'itération que nous étudierons de manière plus détaillée en cours. Il est donc réservé aux étudiants ayant déjà des connaissances de ces notions. Il vous est demandé de modifier votre programme de conversion afin de permettre :

- une utilisation en convertisseur franc → euro ou au contraire euro → franc, suivant un choix exprimé par l'utilisateur au clavier,
- une utilisation en continu du convertisseur, jusqu'à ce que l'utilisateur exprime au clavier le désir d'arrêter le programme (réponse 'o' à la question « voulez-vous sortir [o/n]').

TP 2 — Conditionnelle et Itération

Problème 1 — Un petit peu d'arithmétique

Nous avons vu en travaux dirigés comment multiplier des nombres entiers à l'aide uniquement de l'addition.

1 — Enregistrez ce programme dans un fichier MULTADD.PAS et testez son bon fonctionnement : n'observez-vous pas des résultats bizarres pour certains multiplicandes ? Comment expliquez-vous ces erreurs ?

2 — Modifiez votre programme pour éviter ces effets de bords en forçant l'utilisateur à entrer des valeurs acceptables.

3 — Si la boucle d'itération porte sur le second multiplicande, une opération telle que 2 * 564, pourtant très simple, nécessitera 564 additions. En partant de cette observation, modifiez votre programme pour le rendre plus efficace en temps de calcul.

4 — Après l'addition, vous avez droit désormais d'utiliser la multiplication. En n'utilisant que cette opération, écrire un programme Pascal qui donne la puissance N-ième, avec N entier positif, d'un entier relatif. Le programme ne cherchera pas à éviter les effets de bords dans ce cas (bien qu'il serait facile de le faire en travaillant sur les valeurs des mantisses…).

Remarque : il n'existe pas de fonction puissance prédéfinie dans Turbo Pascal. Pour calculer une puissance sans avoir à utiliser un programme aussi complexe, vous devez utilisez la fonction prédéfinie exponentielle (`exp`).

Problème 2 — Le jeu du "*plus ou moins*" : recherche d'un nombre par dichotomie

Le jeu du "*plus ou moins*" a longtemps fait le bonheur des émissions de jeux des radios dites « périphériques » (c'est à dire à l'époque RTL, Europe 1 ou RMC) non contrôlées[1] par le gouvernement français. La « Valise RLT », qui a donné lieu à un sketch de Jean-Marie Bigard, reposait sur ce jeu au principe très simple : l'animateur (un dénommé Fabrice en l'occurrence…) choisit un nombre — compris généralement entre 1 et 10000 — représentant une somme d'argent (en francs…) cachée dans une valise. Chaque auditeur appelé doit donner une estimation. L'animateur précise à chaque fois si le nombre recherché est supérieur ou inférieur à cette estimation, jusqu'à ce que le gagnant donne enfin la bonne estimation. Si les auditeurs jouent intelligemment, la recherche s'opère par réductions successives de l'intervalle d'encadrement [minimum, maximum] du nombre. L'auditeur qui trouve le premier la bonne valeur emporte alors la cagnotte.

1 — Analyse de problème : formalisez de manière itérative cet algorithme de recherche de la bonne valeur par dichotomie. La saisie de la réponse d'un auditeur correspondra à une lecture au clavier.

2 — Ecrire le programme correspondant et testez son bon fonctionnement. Une session de jeu correspondra à la succession d'affichages suivants :

```
Entrez la valeur que devra trouver l'auditeur : 5340
-------------------------------------------------

Debut du jeu

Entrez une valeur : 2500
La valeur est trop basse
Entrez une valeur : 7500
La valeur est trop élevée
Entre une valeur : 5340

Bravo, c'est la bonne réponse.
Voulez-vous recommencer un nouveau jeu [o/n] : o

Entrez la valeur que devra trouver l'auditeur : 7777

…
```

[1] Ces stations émettaient en dehors du territoire française (Luxembourg, Suisse, Monaco) est étaient donc les seules radios captables en France à ne pas être contrôlées par un Ministère de l'Information qui surveillaient les paroles de tous les intervenants des radios contrôlées par l'Etat (l'ORTF). Ce Ministère fut supprimé après la mort du Général de Gaule, mais il fallut attendre l'arrivée au pouvoir de François Mitterand, en 1981, pour que des radios (alors appelées « radio libres ») puissent enfin émettre en toute liberté depuis le territoire français. La liberté d'expression ne fut donc réelle, en France, que 8 ans avant la chute du Mur de Berlin…

```
Voulez-vous recommencer un nouveau jeu [o/n] : n
Au revoir et merci d'avoir joué à ce jeu passionnant !
```

Problème 3 — Calcul de la racine carrée par la méthode de Newton

Isaac Newton (1642-1727) n'a pas fait que révolutionner notre vision du monde avec sa théorie de la gravitation universelle qui lui serait venue, selon une légende inventée par le dessinateur Gotlib, en recevant sur la tête une pomme tombé de l'arbre sous lequel il se reposait… Parmi ces multiples travaux en mathématiques, Newton a proposé une méthode de calcul par approximation de la racine carrée d'un nombre. Pour tout nombre v dont on cherche la racine carrée, cette méthode repose sur la définition de la suite suivante :

- $U_0 = v_0$ avec v_0 quelconque tel que $v_0 > \sqrt{v} > 1$
- $U_{n+1} = ½ (U_n + v / U_n)$ pour tout entier $n > 0$

On montre aisément que cette suite est toujours bornée (borne inférieure) par $\sqrt{v}$ et qu'elle converge vers cette valeur.

1 — Analyse de problème : formalisez de manière itérative cet algorithme de recherche de la racine carrée.

2 — Ecrire le programme correspondant de calcul de la racine carrée. Celui-ci demandera à l'utilisateur de saisir un nombre, puis calculera les valeurs successives de la suite jusqu'à son $20^{ème}$ terme, et fera au final la comparaison entre la valeur obtenue et celle calculée par la fonction Pascal `sqrt`.

3 — Etudiez l'influence du terme initial v_0 sur la convergence de la suite.

TP 3 — Vecteurs et chaînes de caractères

Problème 1 — Crible d'Eratosthène

Simple problème a priori, la recherche des nombres premiers a gouverné une grande partie des recherches en mathématiques depuis l'antiquité et a eu de nombreuses applications initialement insoupçonnées. Qu'on pense par exemple à leur utilisation dans les techniques modernes de cryptographie. C'est à Eratosthène (III° siècle avant J.C), connu avant tout pour avoir été le premier à fournir une estimation très précise pour l'époque du rayon terrestre (les grecs savaient en effet depuis le V° siècle avant J.C. que la Terre n'était pas plate) que l'on doit l'une des méthodes les plus anciennes de recherche systématique de nombre premiers. Sa méthode, appelée crible d'Eratosthène, consiste à éliminer progressivement tous les multiples d'entiers croissants en partant de 2.

Par exemple, si on cherche les nombres premiers compris entre 1 et N, on va tout d'abord ôter les nombres pairs (multiples de 2), puis les multiples de 3 et ainsi de suite jusqu'à …. un certain nombre que dont nous vous laissons deviner la valeur.

En utilisant un tableau de 100 entiers, écrire un programme Pascal qui affiche à l'écran l'ensemble des nombres premiers compris entre 0 et 100. On prendra garde de faire une analyse complète du problème avant de se lancer dans la programmation à proprement parler.

Problème 2 — Code de César

Employé selon Suétone par César pour sa correspondance avec Cicéron, le code qui porte son nom est la méthode de cryptographie la plus ancienne connue par l'histoire. Il consiste en simple décalage dans l'alphabet des lettres du message à chiffrer. Par exemple, si on effectue un décalage de 3 lettres vers la droite, on remplace A par D, on remplace B par E …

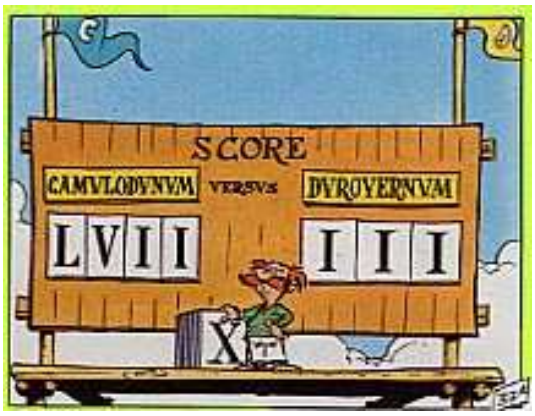
Lettre initiale	A	B	…	W	X	Y	Z
Lettre chiffrée	D	E	…	Z	A	B	C

Ce code très sommaire est bien entendu assez facile à décoder. Pourtant, son extrême simplicité fit qu'il était encore utilisé, pour des messages peu importants, pendant la guerre de Sécession américaine et par l'armée russe en 1915 !

Dans ce problème, nous allons réaliser un codeur / décodeur utilisant précisément le code de César.

1) Réaliser un programme Pascal qui assure le chiffrage d'un texte saisi au clavier. On demandera à chaque utilisation le pas de décalage (positif ou négatif) requis pour le chiffrage et on affichera en le message codé à l'écran.
2) Modifier ce programme pour qu'il fonctionne aussi bien en codeur qu'en décodeur.

Problème 3 — Ils sont fous ces romains !



Restons quelques temps avec César. Les Romains, qui ont tout de même dominé l'Europe pendant les derniers siècles de l'antiquité, étaient un peuple de formidables ingénieurs capables par exemple de bâtir des canaux de plusieurs centaines de kilomètres de long, chefs d'œuvre d'hydraulique nécessitant des calculs d'inclinaison extrêmement précis pour l'époque. Pourtant, Rome utilisait un système de numération qui ne facilitait pas ces calculs : non seulement l'écriture des nombres en chiffres romains répondait à des règles peu systématiques, mais surtout elle rendait dramatiquement difficile les calculs arithmétiques : une simple addition nécessitait en effet des règles de retenues très complexes. Dans ce problème, nous allons simplement nous intéresser à la conversion des nombres écrits en chiffres arabes en chiffres romains, et réciproquement.

Règles de conversion en chiffre romains

Les chiffres romains se composent à l'aide de 7 lettres de l'alphabet écrites en majuscules. Ces lettres sont :

- **I** qui représente 1.
- **V** qui représente 5.
- **X** qui représente 10.
- **L** qui représente 50.
- **C** qui représente 100.
- **D** qui représente 500.
- **M** qui représente 1000.

De 1 à 3 unités, dizaines ou centaines, les lettres correspondantes sont répétées autant de fois que nécessaire :

- II = 2
- XXX = 30
- C = 100

Pour 4 unités, dizaines ou centaines, on prend la lettre de valeur supérieure (V, L ou D) devant laquelle on place une lettre équivalente à la différence (I, X ou C) :

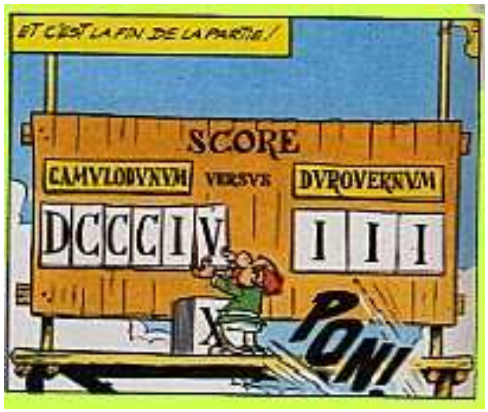
- IV = 1 ôté à 5 = 4
- XL = 10 ôté à 50 = 40
- CD = 100 ôté à 500 = 400

De 6 à 8 unités, dizaines ou centaines, on ajoute le nombre nécessaire (même règle que pour compter de 1 à 3 unités, dizaines ou centaines) :

- VII = 5 + 2 = 7
- LXXX = 50 + 30 = 80
- DC = 500 + 100 = 600

Pour les milliers, la lettre **M** est répétée autant de fois que nécessaire. Exemple : MMMMM = 5000

Pour écrire des chiffres plus complexes, il suffit de commencer par les plus grandes valeurs (milliers) et de décomposer les valeurs en utilisant les règles présentées ci-dessus.



A réaliser

1) Réaliser un programme Pascal qui assure la conversion d'un nombre en chiffres romains. On demandera à l'utilisateur de saisir le nombre au clavier, et on affichera le résultat en réponse.
2) Réaliser un programme Pascal qui réalise la conversion inverse (romain → arabes) en s'interrogeant bien sur la représentation des données en entrée.

TP 4 — Sous-programmes : fonctions et procédures

Problème 1 — Le jeu des dés indiens

Proche du *Jam*, le jeu des *dés indiens* consiste à obtenir, à l'aide de cinq dés, une combinaison la plus forte possible. Chaque joueur à droit au maximum à trois jets, avec possibilité de retirer certains dés après chaque jet. Les combinaisons du jeu de Jam sont les suivantes (par ordre décroissant d'importance) :

la *quinte*	5 faces semblables	(le *jam* du jeu du même nom)
le *carré*	4 faces semblables	
le *full*	un brelan et une paire	
le *brelan*	trois faces semblables	
la *paire*	deux faces semblables	(deux paires valent plus qu'une seule paire)

On suppose qu'après ses trois jets, le joueur ordonne ses dés par valeurs décroissantes. Une paire correspondra alors à une répétition de valeur (par exemple, dans 6 4 3 3 1, 3 est répété une fois), le brelan à deux répétitions et ainsi de suite. Les combinaisons peuvent alors être redéfinies comme suit :

la *quinte*	suite maximale de 5 valeurs constantes
le *carré*	suite maximale de 4 valeurs constantes
le *full*	suite maximale de 3 valeurs constantes et 3 répétitions de valeurs (2 : brelan + 1 : paire)
le *brelan*	suite maximale de 3 valeus constantes et 2 répétitions de valeurs
les *2 paires*	suite maximale de 2 valeurs et 2 répétitions (une par paire)
la *paire*	suite maximale de 2 valeurs et une seule répétition

1 — Ecrire l'algorithme de la fonction qui délivre le nombre de répétitions au sein d'une suite de valeurs.

2 — Ecrire l'algorithme de la fonction qui délivre la taille de la plus longue sous-suite constante au sein d'une suite de 5 valeurs

3 — En utilisant ces fonctions, écrire le programme de détermination des combinaisons aux dés indiens.