GROUPE ESEO

MINI-PROJET LANGAGE C

PUISSANCE 4 Nouvelle Version

Sophie Rousseau
Mickaël Clavreul
Ingénieurs 1ère année

2012 / 2013

Sommaire

INTRODUCTION

Le mini-projet de langage C est l'occasion de mettre en pratique l'ensemble des connaissances accumulées durant les cours/TD/TP d'informatique.

Ce projet consiste en la réalisation d'un programme, en général un jeu, permettant d'aborder un maximum de fonctionnalités et de possibilités qu'offre le langage C.

Dans ce document, sont présentés :

- le projet, le cahier des charges associé et le travail attendu ;
- une présentation de ce qu'il faut faire ou ne pas faire en matière de codage ;
- quelques questions pouvant être utiles au développement et au déboggage du programme.

Nous ne pouvons que vous conseiller plus que fortement de lire l'intégralité du chapitre 1 avant de commencer.

CHAPITRE 1. Le projet : Puissance 4 – Nouvelle Version

1.1 Sujet :

Nous allons programmer/simuler à l'aide d'un programme en Langage C le jeu PUISSANCE 4 Nouvelle Version.

1.1.1 Le principe du jeu

PRESENTATION :

Pourquoi 'Nouvelle Version' ?

Dans le jeu classique, 2 joueurs s'affrontent. Chacun leur tour, ils choisissent une colonne pour un jeton. Celui-ci descend jusqu'à la dernière case libre. Le jeu s'arrête lorsque un des joueurs réussit à aligner 4 jetons (horizontalement, verticalement ou en diagonale) ou lorsque la grille a été remplie sans alignement. Ce dernier cas est une partie nulle.

Dans cette nouvelle version, le jeu propose 2 variantes du Puissance 4 classique :

– Dehors ! → le joueur a la possibilité de soit insérer un pion, soit éjecter un de ses pions. Un puissance 4 gagne;
– Top 10 → on remplit la grille puis on éjecte les pions jusqu'à obtenir 10 pions de sa couleur.

Les règles de jeu de ces 2 variantes sont expliquées dans le paragraphe suivant.

1.1.2 Les règles du jeu

Composition du jeu :

- un plateau de jeu de 6 rangées et 7 colonnes.
- 42 pions : 21 de chaque couleur.

Préparation :

Chaque joueur dispose de 21 pions et la grille est vide.

Déroulement de la partie :

Top 10

Cette version du jeu se déroule en 2 phases :

– le remplissage

Dans cette phase de jeu, chacun leur tour chaque joueur insère un de ses jetons afin de remplir chaque rangée l'une après l'autre.

Attention, le remplissage d'une rangée n'est possible que lorsque la rangée précédente est complète. Sur la figure suivante, dans le cas **a**, Jaune ne peut pas insérer son jeton car la rangée 4 n'est pas complète.

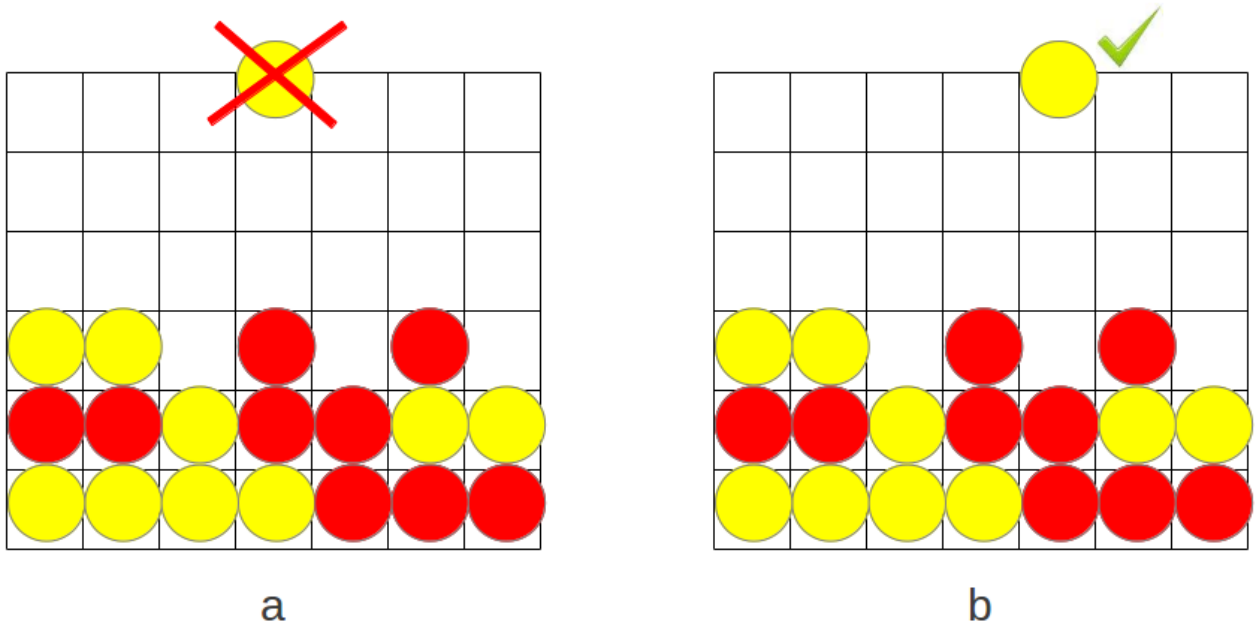


Figure 1. Puissance 4 - Top 10 : Remplissage

– l'éjection.

Une fois la grille remplie, la deuxième phase de jeu commence.

À tour de rôle, chaque joueur peut retirer l'un de ses jetons de la rangée du bas. Si ce pion appartient à un alignement de quatre, il est conservé par le joueur, lui rapporte un point et lui permet de rejouer. Si le pion n'appartient pas à un alignement, il doit être remis en jeu dans une colonne différente.

La figure suivante représente le déroulement d'un tour de jeu dans le cas où Jaune débute la phase d'éjection.

La première figure représente la grille remplie.

Le joueur retire le jeton de la colonne 4 dans la 2e figure. Comme ce jeton appartient à un alignement Puissance 4, le joueur garde le pion, marque un point et peut continuer à jouer.

En retirant ce jeton, il a conservé un puissance 4 en diagonale qui lui permet, dans la figure 3 de retirer et conserver le jeton de la colonne 2.

Comme il vient de retirer un jeton faisant partie d'un puissance 4, il peut de nouveau jouer.

Aucun des 2 jetons restant dans la rangée ne faisant partie d'un puissance 4, il décide d'éjecter celui de la colonne 3 et doit le réinsérer dans le jeu dans une colonne autre que la 3.

Son tour est alors terminé et la main passe au joueur Rouge.

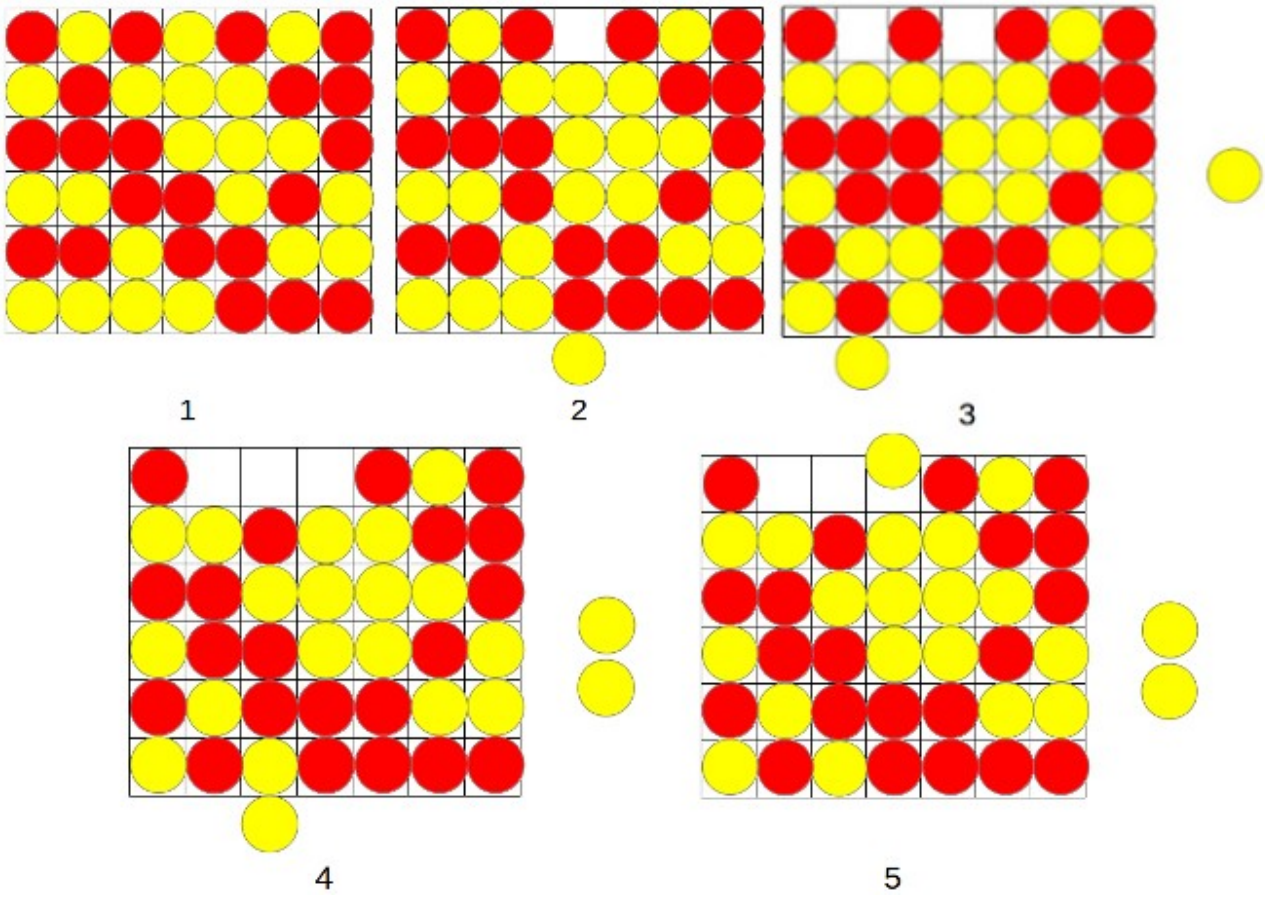


Figure 2. Puissance 4 - Top 10 : Jaune débute la phase d'éjection

Attention, la figure suivante représente un placement interdit pour le dernier jeton éjecté par le joueur jaune.

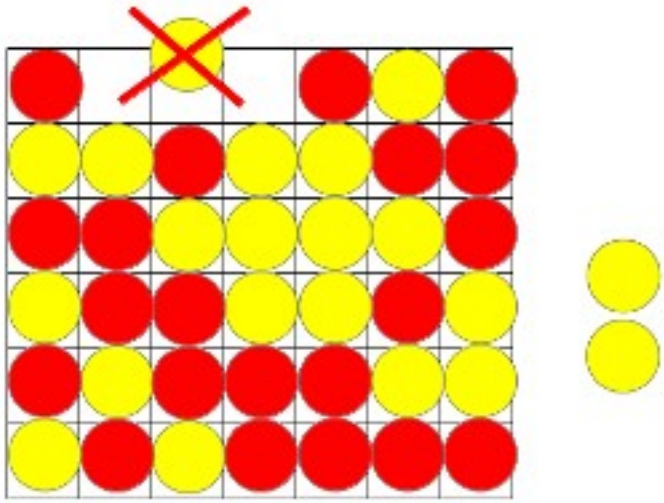


Figure 3. Puissance 4 - Top 10 : réinsertion impossible.

Le seul cas permettant une réinsertion dans la même colonne que celle d'éjection est le cas où aucune autre colonne n'est libre.

Dans le cas où aucun jeton n'est disponible dans la couleur du joueur devant jouer, la main passe à l'adversaire.

Le jeu s'arrête lorsqu'un joueur obtient 10 jetons. Il est alors déclaré vainqueur de la partie.

Dehors !

Dans cette version, les joueurs utilisent la rangée inférieure pour éjecter leurs jetons.

À chaque tour, le joueur a deux possibilités : soit placer normalement un pion en haut d'une colonne, soit éjecter un de ses propres pions placé dans la rangée du bas.

Pour gagner, les joueurs doivent aligner 4 jetons de leur couleur.

Dans la figure ci-dessous, Jaune comme Rouge peuvent gagner si c'est à eux de jouer.

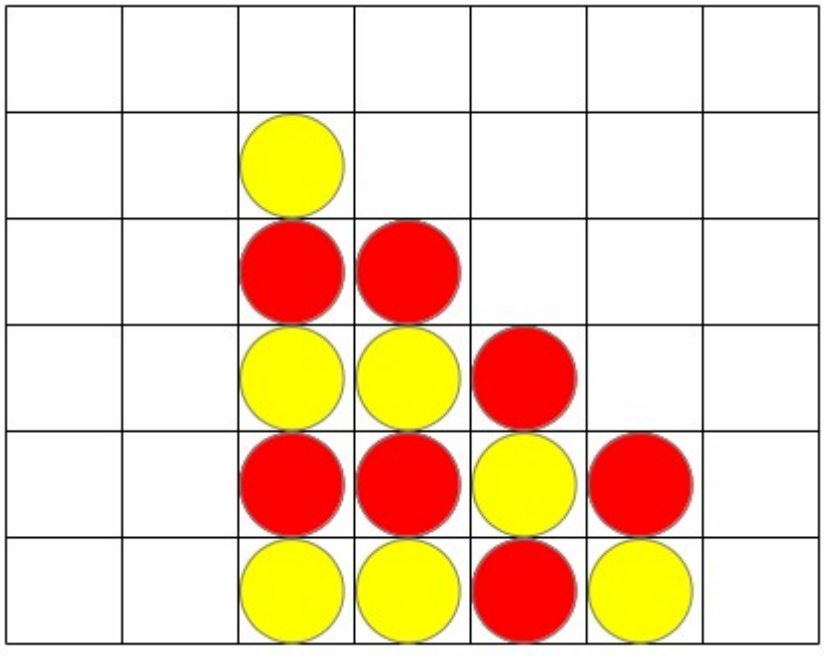


Figure 4. Puissance 4 - Dehors !

Rouge peut gagner de deux façons : soit en jouant dans la colonne 7, soit en éjectant son pion de la colonne 5.

Jaune peut gagner en éjectant son pion de la colonne 3.

1.2 Cahiers des charges :

1.2.1 Cahier des charges de base

On se propose de programmer la version **TOP 10** de ce jeu pour 2 joueurs.

Vous devrez donc gérer la partie entre les 2 joueurs en :

- gérant la partie,
- vérifiant qu'ils ne commettent pas d'erreur,
- détectant la fin de partie.

On vous demande également de pouvoir sauvegarder une partie en cours mais aussi de recharger une partie qui avait été stockée précédemment sur le disque dur.

!!! Faites simple au départ !!!

…combien de binômes avons-nous vu se lancer « tête baissée » voulant gagner du temps au départ et dont le clavier « raisonnait » déjà à peine 1 heure après le sujet distribué mais qui 2 ou 3 séances plus tard ont très très amèrement regretté leur(s) choix !!!

Avec le recul, un choix très anodin ou parfois bien trop compliqué leur a fait perdre des séances entières de 4 heures… Non, non nous n'exagérons pas… !

1.2.2 Cahier des charges (extensions)

A présent, on souhaite que chaque joueur puisse jouer sur son propre écran. Chaque joueur verra donc sur son écran un plateau de jeu lui permettant de suivre l'évolution de son parcours et l'évolution du jeu de son adversaire.

Au niveau de la réalisation informatique, il faut bien comprendre que chacun des binômes va lancer le même programme sur sa propre machine. Il sera donc nécessaire de transférer des informations entre les 2 programmes. Pour réaliser cette communication entre les deux programmes vous pouvez utiliser un fichier qui sera stocké chez l'un des binômes. Il est alors plus simple de vous loguer sur le même compte (sur 2 postes différents…).

Une difficulté peut survenir : en effet, l'un des programmes lit le fichier pendant que l'autre écrit et vice versa, du coup des conflits d'accès peuvent apparaître entre les 2 programmes.

Une première solution consiste à ce que chacun dise à l'autre (oralement) lorsqu'il vient de jouer. Une deuxième solution se trouve en Annexe (Jouer sur 2 ordinateurs différents).

Une dernière remarque, vu que vous lancez 2 fois un programme identique, ce dernier ne sait pas si il joue avec le joueur A ou le joueur B… il vaut donc mieux qu'il vous le demande au départ. Au final, l'idéal serait donc que le programme détecte quand l'autre joueur a joué, et affiche le nouvel état du plateau.

1.2.3 Pour les plus rapides ou les plus forts...:

Au choix :

- Jouer contre l'ordinateur. Dans ce cas, il faut développer une sorte d'intelligence artificielle pour ce dernier...
- Mettre en place, en réutilisant certaines des fonctionnalités réalisées, la version **Dehors !** Et permettre donc aux joueurs de choisir le type de partie.
- Faire du « joli » graphisme. Récupérez les 2 fichiers **bibliotheque.h** et **graphisme.c** sur l'intranet de l'école... et testez les... Le but n'est pas de faire du "super graphisme" ou une gestion compliquée d'événements souris, le langage **Java** qui vous verrez très bientôt se prête bien plus à ce type d'usage...

P.S. : voir la 1ère ligne du fichier graphisme.c avec sa jolie ligne de compilation :

```
gcc graphisme.c -I/usr/X11R6/include/ -o graphisme  -L/usr/X11R6/lib -lX11
```

Faites un copier-coller afin de ne pas vous tromper entre les « 1 » « l »…

- Autre option, pour ceux qui veulent faire du graphisme, utiliser SDL, voici un lien qui peut aider : http://lazyfoo.net/SDL_tutorials/index.php

P.S. : si vous choisissez cette option, il va falloir vous débrouiller seul… les encadrants n'assurent pas de support technique…

- Faire un petit « chat » entre les deux écrans.
- Si vous avez d'autres idées…

1.3 Votre travail

1.3.1 Introduction

Réalisation : en binômes dans l'environnement UNIX.

Ne pas utiliser d'IDE (Environnement de Développement Intégré) de type codeBlocks ou Eclipse.

L'écriture du code se fait à partir de l'éditeur de texte de votre choix (gedit, emacs...) et la compilation et l'exécution du code se fait en ligne de commande dans un terminal.

Autonomie : dans une optique de projet, les professeurs encadrants auront un rôle essentiellement de conseil (choix des structures de données, pistes de réflexion, amélioration du logiciel).

Il est donc de votre ressort de régler les différents bugs de compilation et d'exécution. Ces deux tâches font partie intégrante du travail à fournir sur ce projet.

- une petite « soutenance » avec démonstration de votre programme est prévue lors de la dernière séance de projet. Plus d'informations concernant la soutenance sont données dans la section 1.3.3 de ce document.

1.3.2 Documents à rendre

A rendre **au plus tard à la date qui vous sera communiquée sur le campus numérique dans l'espace dédié au Mini-Projet Langage C**:

- Un **rapport papier** qui contiendra, en plus des éléments attendus dans un rapport de projet (se référer au guide de rédaction de rapport présent sur le campus):
  - les fonctionnalités effectivement réalisées par le logiciel. Insérez un tableau ressemblant à celui-ci dans votre rapport:

	Fonctionnalités	**Oui/Non**	**Commentaires**
	Cahier des charges initial		
	Cahier des charges (extensions)		
	Cahier des charges (plus rapides ou plus forts)		
...			
	Nombre de bugs subsistants :		

  - un tableau qui indique comment vous vous êtes répartis les tâches au sein du binôme,
  - un manuel d'utilisation du logiciel,
  - un descriptif des différents éléments composant votre programme : c'est-à-dire le nom des variables (globales ou non), fonctions et structures de données ainsi que leur rôle,
  - quelques exemples d'exécution (avec au moins 5 copies d'écran (commentées) : voir dans les menus de Ubuntu : *Applications -> Accessoires*),
  - les bugs éventuels (avec une description du fonctionnement attendu)
  - Un listing de tests tel qu'il vous a été présenté dans le cours et le TD de Conception/Tests que vous avez suivi au début de l'année.

- **Sur le campus numérique de l'école**, sur le dépôt correspondant à votre projet (pas de mail accepté), un dossier sous la forme d'une archive **zip** contenant :
  - le **listing du programme** (c'est-à-dire le code de votre programme). Ceci nous permettra d'évaluer la qualité du code et de détecter les éventuels plagiats…
  - un fichier intitulé **readme.txt** qui dit quel(s) fichier(s) il faut compiler et comment lancer le programme.
  - Un **exécutable** de votre programme.
  - La **version PDF du rapport** de projet (un autre format de fichier ne sera pas accepté). Nous précisons aussi que, bien entendu, la version papier et la version PDF du rapport sont identiques.

Il nous faut impérativement un rapport papier. La version numérique seule sans une version papier vous ferait perdre des points.

Attention : Le retard dans la remise des éléments du dossier entraîne des conséquences présentées plus bas.

Les correcteurs seront très sensibles à la clarté du rapport : ainsi, nous souhaitons pouvoir saisir au premier coup d'œil ce que votre programme peut faire et ne pas faire...

De plus, un rapport n'est pas une succession de tableaux ou d'images non structurée.

Un document vous permettant de savoir comment un rapport de projet doit être structuré est présent sur le campus numérique de l'école. À vous de voir si vous souhaitez suivre ou non ces recommandations. Cependant, sachez que certaines parties d'un rapport sont obligatoires. La notation en tiendra compte.

Enfin voici un barème approximatif de notation :

pénalité de retard			
retard (jours)	1	2	3
Pénalité (%)	15,0%	20,0%	30,0%

au delà de 3 jours, la règle conformité = 5 s'applique

pénalité français				
niveau	0	1	2	3
Pénalité (points)	0	1	2	3

	dossier		travail				oral	SOUS	pénalités			TOTAL
NOMS	rapport	code	base	extension	forts	planning		TOTAL	retard (jours)	conformité	français	
0	4	2	6	3		1	4	20	0	0	0	20
0						1	4	20				20

Conformité du dossier

à chaque élément manquant : 20% de la note de pénalité	Rapport format pdf	
	rapport papier	
	code source	
	exécutable	
	readme.txt	

conformité						
niveau	0	1	2	3	4	5
Pénalité (%)	0,0%	20,0%	40,0%	60,0%	80,0%	100,0%

Pénalité de retard : l'heure limite de remise du dossier est 18h00. La pénalité 1 s'applique dès 18h01. Attention, comme précisé sur l'image ci-dessus, au delà de 3 jours de retard, la conformité 5 s'applique, c'est-à-dire que le projet est considéré comme étant non rendu.

Pénalité de français : celle-ci concerne la qualité de langage, l'orthographe, la grammaire, la conjugaison, les formulations de phrases... Elle est exprimée en nombre de points.

Conformité : ceci concerne la composition du dossier à remettre. Comme précisé plus haut dans ce document, le dossier est composé d'un rapport papier et d'un dossier numérique, ce dossier comptant 4 éléments précis. Si un des composants du dossier manque, la case conformité prend une valeur comprise entre 0 et 5. 5 correspondant à la pénalité la plus élevée car elle signifierait que le dossier n'a pas été remis.

Les pénalités exprimées en pourcentage s'appliquent sur la note globale du projet indiquée dans la colonne « sous total ».

Enfin, les correcteurs se réservent le droit de ne pas attribuer une note identique à chaque membre du binôme.

1.3.3 La Soutenance

Déroulement de la soutenance :

- **15 min** ;
- pas de modifications du code pendant la présentation ;
- pas de compilation du code ;
- démonstration réalisée sur les machines de l'école (pas de PC personnel).

Remarque : Cette présentation se déroulant lors de la dernière séance programmée de projet, il se peut que votre jeu ne soit pas terminé. Des dysfonctionnements peuvent exister, il vous faut les connaître et pouvoir les expliquer lors de votre présentation.

Installez-vous et testez le lancement de votre programme une dernière fois au moins 10 minutes avant la soutenance !

10 à 20 % des binômes ont des problèmes de dernière minute…

! Si vous avez une version graphique et une version non graphique, la démonstration commencera par la version non graphique de votre logiciel.

Si vous n'avez qu'une version graphique, il y a un souci : vous n'avez pas respecté le cahier des charges (revoir ce dernier)…

1.3.4 Important :

Pour le « **cahier des charges de base** », tout doit être fait **en mode texte** ! La partie « cahier des charges plus forts » contient la possibilité de faire du graphisme.

Pour l'instant on ne veut pas de graphisme… la raison est simple, le langage JAVA que vous verrez par la suite est bien plus souple pour faire du graphisme que le langage C… **Le graphisme ne sera donc réalisé que par les plus forts**.

Comprenez bien que le client, ne sera pas pleinement satisfait s'il n'a qu'une version graphique… du coup si tel est le cas, vous ne ferez pas le « plein » des points…

Le **cahier des charges de base** est donc **obligatoire**.

C'est un petit projet donc, sauf si vous êtes très très à l'aise, on peut se passer de makefile et de compilation séparée… **Faites simple** !

Dans le cas où vous envisageriez de vous diriger vers la compilation séparée, pensez aux conséquences sur l'implémentation de votre projet, en particulier pour ce qui concerne les variables globales.

Un dernier conseil : évitez la récursivité lorsque son utilisation n'est pas justifiée. Tout au plus dans une fonction très simple, ça peut passer. Dans une grosse fonction même avec des appels récursifs assez simples, l'expérience des années passées montre que les programmes nécessitent bien souvent des heures de debugging, même par les plus chevronnés d'entre vous !

1.3.5 Recommandations :

1.3.5.1 Plus qu'une recommandation, une obligation : étudier au préalable le problème

Comme vous l'avez vu en début d'année, en informatique l'essentiel est dans la conception. Cela veut dire qu'il faut que vous du temps ensemble afin de choisir les variables/fonctions communes que vous allez utiliser. D'autre part, pratiquez la technique du « diviser pour régner » …

Dans l'idéal, le temps d'un projet se répartit en 40% réflexion, 20% codage, 20% test, 20% rédaction de documentations (rapports, manuel d'utilisation, formation des utilisateurs,...)

Reportez-vous au cours et TD de conception/Tests.

De plus, la documentation associée à cette phase de conception devra être présente dans le rapport.

Possibilité d'obtenir un point facilement : sur le campus numérique, dans la section correspondant à votre projet, un dépôt vous permet de nous remettre une version de votre planning prévisionnel avant 18h00 le jour de la première séance de projet.

Ce planning est remis au format PDF et contient :

- la liste des tâches/fonctionnalités que vous aurez identifiées durant votre phase de conception,
- la répartition des tâches entre les membres de l'équipe
- la planification des 5 séances de projet.

Nous insistons sur le fait que ce planning est **prévisionnel** et donc il ne reflétera peut-être pas la réalité de votre travail. C'est pourquoi il sera intéressant de voir figurer dans votre rapport de projet, ce planning prévisionnel ET le planning effectivement réalisé. Ceci vous donnera sûrement matière à discuter dans votre conclusion.

1.3.5.2 Le rapport

Une bonne idée, serait de démarrer le rapport dès la 1ère séance.

1.3.5.3 Sauvegardes

Faites régulièrement des sauvegardes. Par exemple chaque semaine, vous pouvez faire une opération du type :

dupont> **mkdir 20_octobre**

dupont> **cp *.c 20_octobre**

ce qui vous permettra de sauvegarder tous vos sources dans le répertoire 20_octobre.

1.3.5.4 Un conseil...

Nous ne saurions trop insister sur l'importance du mini-projet dans l'acquisition des connaissances. L'informatique s'apprend par la pratique, et il n'y pas d'approche livresque du problème possible. D'éminents académiciens en ont fait l'amère expérience, ne faites pas comme eux. L'informatique ne se bachote pas non plus...

1.3.5.5 Deux remarques

- Les professeurs encadrants sont tenus de vous aider sur les ordinateurs de la « maison » mais pas sur les noteBooks qui sont, de par leur taille bien souvent inconfortables pour travailler à plusieurs dessus.

! Un programme compilé et fonctionnant sur un PC personnel peut ne pas fonctionner /compiler sur les machines « maison »

- Vous devez livrer au moins une version sous LINUX. Une version du soft qui tournerait uniquement sous Windows est insuffisante.

Il est déjà arrivé qu'un binôme s'inspire un peu « trop » du code d'un autre binôme… cela risque de nous crisper… :-< et à être trop crispés, nous risquons de vouloir faire un exemple… exemple qui pourrait aller jusqu'au Conseil de Discipline… A bon entendeur…

Il faut savoir que d'excellents logiciels de détection de plagiat(s) existent, ils sont très performants…

CHAPITRE 2. Programmes illisibles & Convention de nommage

2.1 Programmes illisibles

Il est facile, très facile, de faire des programmes illisibles en C ou en C++. Il existe même un concours du code le plus obscur !

Cela dit, deux choses peuvent être dites à ce propos :

1. Ça n'accroît pas la vitesse du programme. Si l'on veut aller plus vite, il faut revoir l'algorithme ou changer de compilateur (inutile de faire de l'assembleur : les bons compilateurs se débrouillent mieux que les être humains sur ce terrain. L'avantage de l'assembleur est que là, au moins, on est sûr d'avoir un programme illisible.).

2. Ça augmente les chances d'avoir des bogues.

Si vous voulez malgré tout vous amuser, voici quelques conseils utiles :

- écrivez des macros complexes qui font des effets de bords insoupçonnés et qui modifient des variables globales
- abusez de l'opérateur ternaire ?: et surtout de l'opérateur virgule
- utilisez les opérateurs d'incrémentation et de décrémentation à outrance, en version préfixée et suffixée, tout spécialement dans des expressions utilisant des pointeurs
- placez ces opérateurs dans les structures de contrôles. Notamment, utilisez l'opérateur virgule pour faire des instructions composées dans les tests du `while` et dans tous les membres du `for`. Il est souvent possible de mettre le corps du `for` dans les parenthèses
- si nécessaire, utiliser les expressions composées ({ et }) dans les structures de contrôle
- choisissez des noms de variable et de fonction aléatoires (pensez à une phrase, et prenez les premières ou les deuxièmes lettres des mots au hasard)
- regroupez toutes les fonctions dans un même fichier, par ordre de non-appariement
- inversement, dispersez les définitions des variables globales dans tout le programme, si possible dans des fichiers où elles ne sont pas utilisées
- faites des fonctions à rallonge

- ne soignez pas l'apparence de votre programme (pas d'indentation ou, au contraire, trop d'indentations)
- regroupez plusieurs instructions sur une même ligne
- rajoutez des parenthèses là où elles ne sont pas nécessaires
- rajoutez des transtypages là où ils ne sont pas nécessaires
- ne commentez rien, ou mieux, donnez des commentaires sans rapport avec le code.

Exemple Programme illisible

```
/* Que fait ce programme ? */
#include <stdio.h>
int main(void)
   {
int zkmlpf,geikgh,wdxaj;
    scanf("%u",&zkmlpf);for (wdxaj=0,
   geikgh=0;
      ((wdxaj+=++geikgh),geikgh)<zkmlpf;);
   printf("%u",wdxaj); return 0;
}
```

Vous l'aurez compris : il est plus simple de dire ici ce qu'il ne faut pas faire que de dire comment il faut faire.

2.2 Conventions de nommage et commentaires

Dans la plupart des entreprises qui développent des logiciels informatique, les responsables de projets se sont rendu compte que sans des règles de programmation bien définies, il est souvent très difficile d'intervenir dans le code d'un programme écrit par un autre programmeur, ce qui nuit à la productivité de l'entreprise, voire peut la mettre en péril.

Ils utilisent donc des chartes de programmation, destinées à guider les programmeurs dans leur façon de programmer et leurs méthodes de programmation.

Un des exemples les plus simples de ce type de charte est la convention de nommage des variables. Pour cela, on peut utiliser la notation à la hongroise qui préfixe tous les noms de variables par un code pour en indiquer son type. Ce préfixe est toujours écrit en minuscule puis la première lettre suivante en majuscule. On peut étendre ce principe en partant du principe que si les noms des variables comportent plusieurs mots, les lettres seront en minuscules sauf la première lettre de chaque mot, ce qui permet de distinguer rapidement le nom et la signification de chaque variable.

Préfixes	description

b	boolean (booléen true/false)
c	char (caractère)
d	double (float double)
f	float (nombre a virgule flottante)
f	file (fichier)
l	long int (entier long)
n	int (entier)
n	short int (entier court)
p	pointeur
sz	zero-terminated string (chaine de caractères terminée par un char zéro)

Exemples :

```
int nIndex;
char cToucheClavier;
float* pfValeurDebit;
```

Évidemment, il ne faut pas en abuser sinon elle devient inutile.

Ex. : `arpszTableau` pour un tableau de pointeurs de chaînes de caractères.

En ce qui concerne les commentaires, il faut bien sûr en mettre dans le programme, on peut aussi pour augmenter la lisibilité des programmes, ajouter des entêtes pour chaque fonction du programme, décrivant son nom, son utilité, la liste des variables qu'elle reçoit et renvoie, le nom de la personne qui l'a écrite, la liste des différentes modifications et leur date, …

Un exemple d'entête de fonction est présenté dans l'Annexe B- Commenter ses programmes

CHAPITRE 3. LANGAGE C

3.1 Comment faire un tirage aléatoire ?

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

void init_rand() {     /* fonction à appeler une seule fois dès le départ du main() */
      srand(time(NULL));  /* initialise le générateur des nombres pseudoaléatoires  */
}

int tirage_nombre (int inf, int sup) {
   /* renvoie un nombre aléatoire compris entre inf et sup */
   int x;
   if (inf>=sup)
      return inf;

   x=rand()%(sup-inf+1)+inf;
   return x;
}
main () {
   int i;
   init_rand();
   for (i=0;i<20;i++)
      printf("\nTirage aléatoire vaut : %d", tirage_nombre(0,100));
}
```

3.2 Comment mettre de la couleur ?

Cette partie est tirée de l'ouvrage [Rob05] .

Si le terminal utilisé le permet, vous pouvez agrémenter votre programme de quelques attributs : caractères gras, soulignement, couleurs de fond et d'encre inversées, caractères invisibles. Chaque attribut est associé à un nombre.

Pour utiliser cette possibilité, on utilise le code des séquences d'échappement ANSI. On écrit une chaîne de caractères qui débute par "**\E[**" et qui se termine par "**m**". Entre ces 2 expressions, on met un nombre correspondant à l'attribut voulu. Pour utiliser plusieurs attributs, on peut soit utiliser successivement les deux chaînes :

printf("\E[1m"); printf("\E[7m");

soit séparer les nombres par un point virgule :

printf("\E[1;7m");

Le programme suivant **(que vous pouvez tester par copier-coller depuis l'intranet)** utilise la concaténation (c'est-à-dire la mise bout à bout) de chaînes : ainsi, "Hello " "World" est identique à "Hello World" :

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

```
#define NORMAL          "\E[0m"   // Sans attribut
#define BOLD            "\E[1m"   // Gras
#define UNDERLINED      "\E[4m"   // Souligné
#define REVERSEVIDEO    "\E[7m"   // Vidéo inversée
#define CONCEALED       "\E[8m"   // Invisible
#define BOLD_REVERSE    "\E[1;7m" // Cumul: gras + vidéo inversée


int main () {
     printf(BOLD_REVERSE"Hello"NORMAL"\n");
     return 0;
}
```

Enfin, pour gérer les couleurs: la couleur du fond se précise avec la séquence "**\E[... m**", et le code numérique suivant :

Couleur	**BLACK**	**RED**	**GREEN**	**YELLOW**	**BLUE**	**MAGENTA**	**CYAN**	**WHITE**
Caractères	30	31	32	33	34	35	36	37
Fond	40	41	42	43	44	45	46	47

Ainsi, pour écrire en caractères gras verts sur fond rouge, on écrit :

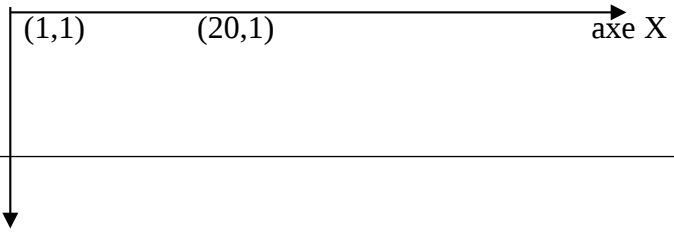
printf("\E[1;32;41m");

3.3 Comment écrire à différents endroits de l'écran ?

Testez le programme suivant (**copier-coller depuis l'intranet**, afin de ne pas tout retaper...).

```
#include <stdio.h>

#define EffacerEcran          system("clear")
#define moveto(x,y)           printf("\033[%d;%dH",y,x)


main() {
   EffacerEcran;

   moveto (1,1);  printf("(1,1)\n");  /* important de placer l'"\n": sera expliqué dans la suite */
   moveto (20,1); printf("(20,1)\n");
   moveto (1,10); printf("(1,10)\n");

   printf("Appuyez sur une touche puis <retour> pour continuer\n");
   scanf("%d",&i);
}
```

Remarque : vous voyez donc que le moveto a ses axes orientés comme ceci:



(1,10)

axe Y

3.4 Un exemple de programme en langage C : le jeu de la vie

Ce programme a pour seul but de vous rafraîchir la mémoire concernant le passage d'un tableau en paramètre par exemple (on se souviendra que seule la première adresse du tableau est passée à la fonction).

Vous pouvez vous inspirer de ce programme !

```
/*************************************
/*             JEU DE LA VIE
/*************************************/

#include <stdio.h>
#include <stdlib.h>
#define TAILLE_SOUS_MATRICE 7      /* On peut avoir 7 * 7 cellules vivantes */
#define TAILLE_SUR_MATRICE  9      /* On place une bordure autour qui facilite la « vie » */
                                   /* au programmeur… */

/***************** PROTOTYPES :**********************/
void Init(int matrice [][TAILLE_SUR_MATRICE ]);
int Nombre_voisins (int matrice [][TAILLE_SUR_MATRICE ], int ligne, int colonne);
void Mise_a_jour(int matrice[][TAILLE_SUR_MATRICE ]);
void Affiche_matrice(int matrice [][TAILLE_SUR_MATRICE ]);
void Ligne(int largeur);
/*************************************/

int main( void ) {
  int i;
  int nbr_cycles;
  int   matrice[TAILLE_SUR_MATRICE] [TAILLE_SUR_MATRICE ];
  char s[2];

 printf("Nombre de cycles : ");
 scanf("%i", &nbr_cycles);

 Init(matrice);
 printf("La population au départ : \n");
 Affiche_matrice(matrice);
 printf("Pressez sur ENTER pour continuer...\n");
 gets(s);

 for(i=0; i<nbr_cycles; i++)  {
    Mise_a_jour (matrice);
    printf("La population après %d cycles : \n", i+1);
    Affiche_matrice (matrice);

    printf("Pressez sur ENTER pour continuer...\n");
    gets(s);
 }
 return 0;
}

/*************************************/          /* Initialisation de la matrice */
void Init(int matrice [][TAILLE_SUR_MATRICE ]) {
/*************************************/
   int i,j;
```

```
   for(i=0; i<TAILLE_SUR_MATRICE ; i++)
     for(j=0; j<TAILLE_SUR_MATRICE ; j++)
        matrice[i][j] = (i<=j && i>0 && j<=7)?1:0;
 }

/*************************************/      /* Calcul du nombre de voisins vivants */
int Nombre_voisins (int matrice [][TAILLE_SUR_MATRICE ], int ligne, int colonne) {
/*************************************/
   return matrice[ligne-1][colonne] +
             matrice[ligne+1][colonne]+
             matrice[ligne][colonne-1] +
             matrice[ligne][colonne+1];
}


/*************************************/          /* Correspond à l'étape n+1 */
void Mise_a_jour(int matrice[ ][TAILLE_SUR_MATRICE ]) {
/*************************************/
 int i,j;
 int nbr_voisins;
 int matrice_densite[TAILLE_SOUS_MATRICE][ TAILLE_SOUS_MATRICE];
 /* matrice qui comptabilise le nombre de voisins */
 /* et cela, case par case */

 for(i=0; i< TAILLE_SOUS_MATRICE; i++)
   for(j=0; j< TAILLE_SOUS_MATRICE; j++)
      matrice_densite[i][j]=Nombre_voisins(matrice,i+1,j+1);
       /* i+1 et j+1 car on passe de la SOUS_MATRICE à la MATRICE   */

 for(i=0; i< TAILLE_SOUS_MATRICE; i++)
  for(j=0; j< TAILLE_SOUS_MATRICE; j++) {
     nbr_voisins=matrice_densite[i][j];
     if(nbr_voisins==2)
        matrice[i+1][j+1]=1;
   else if (nbr_voisins==0 || nbr_voisins==4)
        matrice[i+1][j+1]=0;

 }
}



/*************************************/      /* Affichage à l'écran des cellules vivantes */
void Affiche_matrice(int matrice[ ][TAILLE_SUR_MATRICE ]) {
/*************************************/
 int i,j;

  for(i=1; i<=TAILLE_SOUS_MATRICE; i++) {
      Ligne(7);
      for(j=1; j<= TAILLE_SOUS_MATRICE; j++)
         printf("|%c", matrice[i][j] ?'*':' ');
      printf("|\n");
    }
    Ligne(TAILLE_SOUS_MATRICE);
}


/*************************************/      /* Tracé d'une ligne */
void Ligne(int largeur) {
/*************************************/
  int i;
  for(i=0; i<largeur; i++)
    printf("+-");

 printf("+\n");
}
```

Remarque 1 :

Vous savez que le passage des arguments à une fonction se fait par valeur (pas de modification de cette valeur à la sortie de la fonction).

Remarque 2 :

En lisant par exemple la fonction **void Init(int matrice [][TAILLE_SUR_MATRICE])**, on pourrait se dire que la fonction reçoit la matrice TAILLE_SUR_MATRICE *TAILLE_SUR_MATRICE en entrée et qu'à la sortie elle ne sera pas modifiée, l'instruction **matrice[i][j] = (i<=j && i>0 && j<=7)?1:0;** serait alors vaine. En fait ce qui est passé à la fonction Init n'est autre que l'adresse mémoire de la matrice, en gros **&matrice[0][0]**. Dès lors la fonction ne pourra effectivement pas modifier l'adresse de **matrice[0][0]** par contre elle pourra modifier **matrice[0][0], matrice[1][0]**…

La partie qui suit dresse un panorama des problèmes rencontrés par vos prédécesseurs... **Elle est donc très utile !**

3.5 Les 12 erreurs les plus fréquentes :

Il semble nécessaire de rappeler ici les 12 erreurs les plus fréquentes que l'on peut commettre en écrivant un programme en Langage C :

Erreur	Version correcte
if (i=0)	`if (i==0)`
`scanf ("%d",n) ;`	`scanf ("%d",&n) ;`
`scanf ("%d\n",n) ;`	`scanf ("%d",&n) ;` /* rien d'autre que le(s) format(s) entre les " " ! */
`scanf ("%s",s)` ⇔ `gets(s)`	`gets(s)` permet de lire une phrase complète
if (a & b)	`if (a && b)`
oublier d'initialiser une variable	
tab[i,j]	`tab[i][j]`
les bornes d'un tableau varient entre 1 et N	les bornes d'un tableau varient entre **0** et **N-1**
char c ; `printf ("%s",c) ;`	char c ; `printf ("%c",c)`
char * s ; `printf ("%c",s) ;`	char * s ; `printf ("%s",s)`

`char chaine[3]= "12345" ;`	`char chaine[6]= "12345" ;`
	% on prévoit la place pour l'anti-slash 0
	ou mieux :
	`char chaine[]= "12345" ;`

`char chaine[]='12345' ;`	`char chaine[]= "12345" ;`
si vous utilisez des fonctions mathématiques	compiler le programme par **gcc –o essai essai.c –lm**

telles que **sqrt**, **pow**…	

3.6 *Piège avec le printf et la fonction sleep*

Afin de faire attendre l'ordinateur, il suffit d'utiliser la fonction "`sleep`".

Exemple:

```
printf("*");
sleep(1);
printf("*");
sleep(1);
```

! Il est fort probable que ce programme n'affiche rien ?!

Préférez-lui le programme suivant avec le "\n":

```
printf("*\n");
sleep(1);
printf("*\n");
sleep(1);
```

La raison est la suivante : le compilateur C optimise ses entrées-sorties. Dès lors un programme n'affichera parfois ses données à l'écran que lorsqu'il rencontre un "\n". Tant qu'il n'y a pas d' "\n", le programme conserve ce qu'il doit afficher dans une zone tampon et ne vide ce tampon que lorsqu'il tombe sur un "\n".

Enfin, une autre solution :

```
printf("*"); fflush(stdout);
sleep(1);
printf("*"); fflush(stdout);
sleep(1);
```

3.7 *Avez-vous bien compris ceci ?!?*

Soit le programme suivant:


```
#include <stdio.h>

main() {
     int i;              /* Ligne 0 */
     printf ("%d\n",i);  /* Ligne 1 */

     int * p;            /* Ligne 2 */
     * p = 12;           /* Ligne 3 */
}
```

A la ligne 0, le Langage C se fera un plaisir de vous afficher n'importe quoi car la variable **i** n'a pas été initialisée.

La dernière ligne a de fortes chances de planter le programme! La raison en est que **p** n'a pas été initialisée non plus. En particulier si on faisait un **printf("%p",p)**, on pourrait obtenir n'importe quoi ! Car p, n'est qu'un pointeur, c'est-à-dire une variable contenant une valeur. Bien entendu, dans la pratique, cette valeur n'est pas quelconque: elle désigne bien souvent l'adresse d'une variable.

Donc le plantage sera dû au fait que **p** pointe n'importe où et que vous essayez d'initialiser ce n'importe où.

C'est important d'avoir bien compris ceci pour poursuivre en I2...

3.8 Un petit exemple pour vous rafraîchir la mémoire sur les pointeurs

Pour bien comprendre les pointeurs et leur arithmétique, essayez le programme suivant :

```
#include <stdio.h>
main() {
     int A[5]={2,3,-1,20,3} ,  *p ;

      printf("%p\n",A) ;
      p=A ;
      printf("%p\n",p+2) ;
      printf("%d\n",*(p+2)) ;
      printf("%p\n",*p+2) ;
      return(0) ;
}
```

3.9 Je souhaite modifier un ou plusieurs arguments passés à ma fonction

Rappel très important !!!

Vous savez qu'une fonction n'est pas capable de modifier ses arguments :

Exemple :

```
void double (int p){
   p=p+p ;
}

main () {
   int i =2;
   double(i) ;
   printf("i vaut à présent :%d",i);   /* il vaut toujours 2 !!!
*/
```

```
}
```

La solution consiste donc à faire :

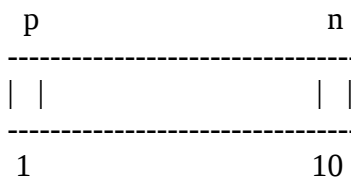
```
void double (int * p){
   *p =  *p +  *p ;
}

main () {
   int i =2;
   double(&i) ;
   printf("i vaut à présent :%d",i);   /* i  vaut bien 4 !!! */
}
```

3.10 J'ai encore du mal avec les pointeurs

A présent, étudions le programme suivant dont le but est simplement de modifier la valeur de N:

```
#include <stdio.h>
void saisie (int *P);

main() {
     int n;
     saisie(&n);
}
void saisie (int *p) {
     printf("Entrez un nombre:\n");
     scanf ("%d",p);
}
```

Imaginons que **p** se trouve en mémoire logé à l'adresse 1 et **n** à l'adresse 10 :



A présent, déroulons le programme principal :

1. `int n;` *déclaration de n : n vaut n'importe quoi vu qu'il n'a pas été initialisé !*
2. **`saisie`**`(&n);` *appel de la fonction saisie, avec la valeur 10 (l'adresse de n)*
3. `void` **`saisie`** `(int *p)` *p vaut donc 10*
4. `printf("Entrez un nombre:\n");`
5. `scanf (p);` *la fonction scanf va stocker ce que l'utilisateur tape au clavier à partir de l'adresse mémoire 10*

3.11 Problèmes avec le *scanf*

Pour mieux comprendre, prenons le programme suivant que vous pouvez/devez tester (vous pouvez récupérer la version intranet)

```
#include <stdio.h>

main () {
       int i;
       int i1,i2 ;
       char c1,c2;

       printf("1) Entrez un nombre: ");
       scanf("%d",&i1);
       printf("2) Entrez un nombre: ");
       scanf("%d",&i2);

       printf("1) Entrez une lettre: ");
       scanf("%c",&c1);
       printf("2) Entrez une lettre: ");
       scanf("%c",&c2);

       printf("1) J'ai récupéré lettre 1:%d\n",(int)c1);    % renverra 10   ⇔ code
                                                             % ascii de '\n'
       printf("2) J'ai récupéré lettre 2:%d\n",(int)c2);
}
```

On voit donc que l'anti-slash n subsiste du premier caractère et pollue la variable `c2` (en effet, quand vous faites un `scanf ("%c",&c1)`, vous demandez à l'ordinateur de lire un unique caractère. Du coup l'"\n" restera de côté pour l'instant). Une solution pour remédier à ce travers consiste à utiliser systématiquement la fonction `gets` (chaine) à la place de chaque `scanf`. Lisez également les paragraphes suivants qui pourront vous servir.

Une autre solution, vue l'année dernière serait celle-ci :

```
printf("1) Entrez un nombre: ");
scanf("%d",&i1);
while (getchar()!='\n')
      ;
printf("2) Entrez un nombre: ");
scanf("%d",&i2);
```

Conclusion: en théorie, 2 solutions sont possibles:

- plus de `scanf`, ou alors utilisez la solution avec `getchar()`
- remplacer tous les `scanf ("%d", ....` `scanf("%c",....)` par du `gets`

Cette dernière solution est normalement celle qui vous fera gagner le plus de temps.

```
/* ancienne version: */
   int i;
   char c;
   printf("Entrez un caractère:");
   scanf ("%c",&c);
   printf("Entrez un chiffre:");
   scanf("%d",&i);

/* nouvelle version: */
   int i;
   char c;
   char chaine[100];

   printf("Entrez un caractère:");
   gets(chaine);
   c=chaine[0];

   printf("Entrez un chiffre:");
   gets(chaine);
   sscanf(chaine,"%d",&i) ;
```

3.12 Quand j'utilise la fonction *gets*, j'obtiens systématiquement des warnings

En première lecture, on pourrait dire que c'est « normal ».

Prenons un exemple :

```
char chaine[10] ;
gets(chaine) ;
```

Supposons que votre programme soit utilisé sur internet et qu'un utilisateur malveillant entre une chaîne de caractères plus grande que 10 caractères, par exemple : `"ABCDEFGHIJKLMONPQR"`. Dans ce cas, à l'exécution, votre programme va recopier à partir de l'adresse de la variable `chaine` les caractères A, B, …. Dès lors, les zones mémoires qui suivent la variable `chaine` seront écrasées. Ceci explique que le compilateur vous indique un warning…

Pour y remédier, vous pouvez utiliser:

```
fgets(buffer, 128, stdin);  /* on lit au maximum 128 sur le fichier
                               stdin qui désigne le
                               fichier attaché au clavier en fait */
```

Remarque: *fgets* présente un petit défaut, elle vous place un '\n' dans ce qui a été lu. Une solution est présentée dans le paragraphe suivant pour y remédier.

3.13 Comment convertir une chaîne de caractères contenant un chiffre en un *int* ?

```
int i ;
char chaine[10] ;  // dimensionnée assez grande !
gets(chaine) ;
i=chaine[0]-'0' ;  /*  va convertir le premier caractère de cette chaîne en un
```

```
                        entier */
```

3.14 Prototypes et warnings

Rappelez-vous qu'il est plus prudent de placer tous les prototypes de fonctions que vous appelez juste en dessous de la suite de #include. Un prototype représente l'entête d'une fonction.

Exemple en gras ci-dessous:

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

float mystere (int i) ;   // prototype de la fonction mystere     (ligne A)

main () {
      int j;
      printf("Entrez une valeur:");
      scanf("%d",&j);
      printf("Résultat de la fonction mystere:%f\n",mystere(j)); // (ligne B)
}

float mystere (int i) {
      return i*i;
}
```

Si vous omettez le prototype, au moment où le programme rencontre l'appel à la fonction mystere (ligne B), le compilateur rencontre pour la première fois cette fonction. Or il ne sait pas ce que fait cette fonction et a fortiori il ignore le type du type du résultat qui sera renvoyé. Dans ce cas, le compilateur suppose à tort que ce résultat sera du type `int` ce qui risque de poser des problèmes dans la suite…

3.15 J'ai des warnings à la compilation

Il est important de comprendre qu'en langage C, un warning équivaut à une "erreur larvée". Ainsi, vous pouvez effectivement exécuter votre programme et ce malgré les warnings.

Le problème c'est que vous risquez de "payer" cela par la suite.

Exemple:

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

/*************************************/
void Affiche_matrice(int matrice[9][9]);       / *prototype */
/*************************************/
 ...

main() {
      int i;
      int matrice[9][9];
      ...
      Affiche_matrice(matrice[9][9]);          /* ligne A */
}

...

/*************************************/
void Affiche_matrice(int matrice[9][9]) {
/*************************************/
```

```
 int i,j;

 for(i=1; i<=7; i++) {
     for(j=1; j<=7; j++)
           printf("|%c", matrice[i][j] ?'*':' ');
     printf("|\n");
    }
}
```

La ligne A vous renverra un warning... Pourquoi ? Prenons le cas où la ligne A serait remplacée par:

```
Affiche_matrice(matrice[1][1]);          /* ligne B */
```

Dès lors, on voit bien le problème, le ligne B ne passe pas tout le tableau matrice à la fonction **`Affiche_matrice`**, mais uniquement la case `[1][1]`. La solution consisterait donc à écrire:

```
Affiche_matrice(matrice);          /* ligne C */
```

Conclusion: considérez tous les warnings comme des erreurs et éliminez les (cela ne concerne cependant pas les warnings provenant de l'usage de `gets`).

3.16 J'utilise la fonction pow ou sqrt ou cos ou sin et je n'arrive pas à compiler mon programme

- ne pas oublier le " **`#include <math.h>`** "
- au lieu de compiler votre programme par : **`gcc –o essai essai.c`**

  faites : **`gcc –o essai essai.c -lm`**

ceci a pour effet de compiler le programme en invitant le compilateur à rechercher la bibliothèque mathématique.

3.17 Je reçois l'une des erreurs : "segmentation fault" ou "core dump"

Ce type d'erreur vous indique que le programme a probablement accédé à une zone mémoire à laquelle il n'aurait pas dû… Voici deux erreurs classiques :

Exemple 1: un oubli du « & » pour le **`scanf`** peut produire ce type d'erreur :

```
…
int i ;
scanf ("%d",i);     /* ???!! */
…
```

Exemple 2 : un accès en dehors des limites d'un tableau est déconseillée :

…

```
int Tab[12];
Tab[15]=120 ;   /* ?!? */
…
```

Pour savoir d'où provient votre erreur, utilisez **ddd** :

1°) compiler avec l'option –g : gcc –o essai essai.c **–g**

2°) ddd essai

3°) cliquez sur Run (il faut peut-être chercher un peu dans les menus où il se trouve…)

D'autres préfèreront peut-être utiliser des mouchards jusqu'à voir d'où provient l'erreur précisément. On place alors des **`printf`** à certains endroits stratégiques :

```
…
int i ;
…
printf ("Je suis là,  i vaut:%d\n",i);
…
```

Si vous choisissez "l'option mouchards", pensez à terminer vos **`printf`** par des "\n".

La raison est la suivante : le compilateur C optimise ses entrées-sorties. Dès lors un programme n'affichera parfois ses données à l'écran que lorsqu'il rencontre un "\n". Tant qu'il n'y a pas d' "\n", le programme conserve ce qu'il doit afficher dans une zone tampon et ne vide ce tampon que lorsqu'il tombe sur un "\".

Exemple:

```
...
printf("Début des mouchards\n");
printf ("Je suis là,  i vaut:%d",i);  /* ligne A */
printf ("Je suis là,  j vaut:%d",j);  /* ligne B */
...
<<<prenons le cas où votre programme s'arrête ici>>>
...
printf ("\n");  /* ligne Z */
```

sortie écran:

```
Début des mouchards

bash.01>
```

Vous pourriez en conclure (à tort), que le programme n'a pas atteint la ligne A alors que la chaîne de caractères `"Je suis là,  i vaut:%d"` se trouve dans un tampon mémoire.

Dans ce cas, il est plus prudent d'écrire:

```
printf("Début des mouchards\n");
printf ("Je suis là,  i vaut:%d\n",i);  /* ligne A */
printf ("Je suis là,  j vaut:%d\n",j);  /* ligne B */
...
```

3.18 Longueur des fonctions

Écrivez de petites fonctions: **en qualité logicielle**, une fonction ne doit pas dépasser une page d'écran d'ordinateur, c'est-à-dire 25 lignes!

3.19 Afficher le contenu d'un fichier texte à l'écran

Pour ce faire, on utilise la fonction **fgets**:

```
char *fgets (char *ligne, int maxligne, FILE *p_fichier)
```

fgets lit la ligne suivante du fichier associé à **p_fichier** (y compris le caractère de fin de ligne) et le stocke dans le tableau de caractères **ligne**. Cette fonction lira au maximum **maxligne-1** caractères.

Voici le listing du programme qui fait cette opération:

```
#include <stdio.h>

const maxligne=100;
char ligne[100];
FILE *p_fichier;

main() {
p_fichier=fopen("ListePoints","r");

while (! feof(p_fichier)) {
            fgets(ligne,maxligne,p_fichier);
      if (!feof(p_fichier)) printf("J'ai lu:%s\n",ligne);
}

fclose(p_fichier);
}
```

3.20 Comment récupérer l'heure ?

Le mieux est sans doute que vous testiez le programme suivant par copier-coller:

```
#include <time.h>

/* renvoie le nombre de secondes de l'heure en cours */
      int Nbr_secondes() {
      char date[30];
      char jour_semaine[5],mois[5];
      int jour,heure,minutes,secondes,annee;
      time_t temps;

      time(&temps);
```

```
      strcpy(date,ctime(&temps));
      printf("\ndate=%s\n",date);
              sscanf(date,"%s%s%d%d:%d:%d
      %d",jour_semaine,mois,&jour,&heure,&minutes,
      &secondes,&annee);

      return secondes;
}

main () {
      printf("\nsecondes=%d\n",Nbr_secondes());
}
```

3.21 Les goto(s) ?!

En théorie, vous ne les avez pas vus en P1-P2, c'est très bien ainsi. Pour ceux qui ne connaissent pas, cela permet de sauter d'un endroit à l'autre du programme. On assimile cela à de la programmation spaghettis… donc n'en utilisez pas, ça pourrait crisper les correcteurs…

3.22 Pour finir…

Une récompense pour ceux qui sont arrivés jusque là ;-) La fonction `sscanf` fait des miracles, voici un exemple pour vous rafraîchir la mémoire :

```
 char s[]=,"12.5 12.3 11.6";     /* extraction de 3 nombres d'une chaîne */
float a,b,c;
int code ;

code=sscanf(s,"%f%f%f",&a,&b,&c) ;
```

`sscanf` renvoie une valeur négative en cas d'erreur, et est égal au nombre de variables affectées sinon.

CONCLUSION

Vous devriez avoir toutes les informations nécessaires au développement de votre application.

Nous insistons cependant sur l'importance de la phase de conception qui ne pourra que vous rendre service.

A vous de jouer.

Toujours soucieux d'être à votre écoute, nous sommes à la recherche de sujets de projets (éventuellement) plus intéressants/innovants.

Aussi, si vous avez une idée à proposer veuillez nous l'exposer en quelques lignes (le jeu d'échecs, jeu de dames, abalone ont déjà été réalisés...).

BIBLIOGRAPHIE

[Rob05] **Philippe ROBINET**. *Le langage C par l'exemple,* éditions Ellipse – 2005

[Rou10] *Comment rédiger un rapport ESEO – Rapport de projet – 2010*

[Ber et Sch 2010] **Berthomier E., Schang D.** *Le C en 20 heures.* Editions ILV-Framabook – 2010

ANNEXE A. Jouer sur 2 ordinateurs différents

Vous pouvez tester le programme suivant (récupérez le sur le campus numérique : **2joueurs.c**) sur 2 postes différents ou dans l'immédiat, sur le même ordinateur mais en ouvrant 2 fenêtres…

```
#include <stdio.h>

void initialisation_2_joueurs ();

void joueur_1_a_fini_de_jouer() ;
void joueur_2_a_fini_de_jouer() ;

void joueur_1_attend_que_joueur_2_a_fini_de_jouer() ;
void joueur_2_attend_que_joueur_1_a_fini_de_jouer() ;

void initialisation_2_joueurs () {
      system("rm joueur_1_a_joue 2> /dev/null"); /* sera explique ci-dessous */
      system("rm joueur_2_a_joue 2> /dev/null");
}
void joueur_1_a_fini_de_jouer() {
      FILE * joueur_1_a_joue ;
      joueur_1_a_joue=fopen("joueur_1_a_joue","w");
      fclose(joueur_1_a_joue);
}
void joueur_2_a_fini_de_jouer() {
      FILE * joueur_2_a_joue ;
      joueur_2_a_joue=fopen("joueur_2_a_joue","w");
      fclose(joueur_2_a_joue);
}
void joueur_1_attend_que_joueur_2_a_joue() {
      FILE * joueur_2_a_joue ;
      joueur_2_a_joue=NULL ;
      while (joueur_2_a_joue==NULL) {
            system("ls -l > /dev/null");   /* commande UNIX : voir ci-dessous */
                                           /* le detail de cette commande*/
            joueur_2_a_joue=fopen("joueur_2_a_joue","r");
            usleep(100000);                          /* on freine un peu... */
      }
      fclose(joueur_2_a_joue);
      system("rm joueur_2_a_joue 2> /dev/null");
```

```
}
void joueur_2_attend_que_joueur_1_a_joue() {
      FILE * joueur_1_a_joue ;
      joueur_1_a_joue=NULL ;
      while (joueur_1_a_joue==NULL) {
            system("ls -l > /dev/null"); /* commande UNIX : voir ci-dessous */
                                         /* le detail de cette commande       */
            joueur_1_a_joue=fopen("joueur_1_a_joue","r");
            usleep(100000);                 /* on freine un peu... */
      }
      fclose(joueur_1_a_joue);
      system("rm joueur_1_a_joue 2> /dev/null");
}
main () {
      char numero_joueur[2];
      char chaine[2];
      printf("\nQuel joueur etes vous (1/2) ?\n");
      gets(numero_joueur);

      initialisation_2_joueurs ();  // !!! ne pas oublier !!!
      while (1) {
            if (numero_joueur[0]=='1') {
                  printf("\n\n\nJoueur 1 va jouer...\n");
                  printf("   Veuillez appuyer sur une touche...");
                  gets(chaine);
                  printf("Joueur 1 a fini de jouer...\n");
                  joueur_1_a_fini_de_jouer () ;
                  printf("\nJoueur 1 attend que Joueur 2 a joue...\n");
                  joueur_1_attend_que_joueur_2_a_joue() ;
            }else{
                  printf("\n\n\nJoueur 2 attend que Joueur 1 a joue...\n\n");
                  joueur_2_attend_que_joueur_1_a_joue() ;
                  printf("Joueur 2 va jouer...\n");
                  printf("   Veuillez appuyer sur une touche...");
                  gets(chaine);
                  printf("Joueur 2 a fini de jouer...\n");
                  joueur_2_a_fini_de_jouer () ;
            }
      }
}
```

Plutôt que d'expliquer tout de suite la commande `system("rm joueur_1_a_joue 2> /dev/null");` nous allons détailler tout d'abord : `system("ls -l > /dev/null");`

- la commande `system` permet d'appeler une commande UNIX à partir d'un programme C. Donc, si on faisait `system("ls -l")`, cela permettrait de lister le contenu du répertoire courant et ce, directement à partir de notre programme C.

- la commande `system("ls -l > mon_fichier")` permettrait de ne plus faire « atterrir » le résultat du `ls -l` à l'écran mais dans le fichier `mon_fichier`. On parle de redirection… on en reparlera en I2…

- le répertoire `/dev` est un répertoire qui appartient à UNIX

- `/dev/null` est un fichier « poubelle » qui appartient à UNIX

- au final, l'instruction `system("ls -l > /dev/null")` liste le contenu de votre répertoire courant et le stocke dans le fichier « poubelle » `/dev/null`

- La raison de faire ceci (quand on travaille à 2 personnes sur le même compte mais sur 2 postes différents) est d'obliger l'ordinateur à bien mettre tous ses caches des disques. En effet, prenons le cas où le binôme 1 modifie le fichier `joueur_1_a_joue` sur son ordinateur par un `joueur_1_a_joue=fopen("joueur_1_a_joue","w");` L'expérience a montré que si le binôme 2, sur un second ordinateur (mais connecté sur le même compte) fait une opération du type `joueur_1_a_joue=fopen("joueur_1_a_joue", "r");` alors 45 secondes peuvent s'écouler avant que ce fichier ne soit accessible/visible sur le second poste !

La commande `system("rm joueur_1_a_joue");` fonctionne sur la même base. Le problème vient lorsque le fichier `joueur_1_a_joue` n'existe pas, dans ce cas, vous avez un joli message d'erreur du type « `rm ne peut enlever 'joueur_1_a_joue' : aucun fichier de ce type` ». On pourrait faire alors `system("rm joueur_1_a_joue > /dev/null");`. Nous reviendrons en I2, là-dessus, ça ne fonctionne pas. En fait il faut faire `system("rm joueur_1_a_joue 2> /dev/null");` car justement on cherche à rediriger le canal des erreurs (qui est redirigé par 2>)… c'est du détail, si vous n'avez pas tout saisi à 100%, ce n'est pas grave, découverte/révision en I2…

Enfin, pour informer le joueur 1, que le joueur 2 vient de jouer, on pourrait créer un second fichier et procéder de manière symétrique :

```
FILE *joueur_2_a_joue;     /* ce fichier va servir à informer le joueur 1 que */
                           /* le joueur 2 vient de jouer */
joueur_2_a_joue =NULL;
```

Il ne vous reste plus qu'à adapter ce programme… après mûre réflexion… Un conseil ne vous « jetez pas sur le clavier… » croyez en notre expérience, une réflexion assez longue au départ peut vous faire gagner des heures par la suite…

Piège : méfiez-vous d'une chose, lorsque vous faites :

```
fclose(joueur_1_a_joue);
```

La valeur de `joueur_1_a_joue` est au mieux `NULL`, au pire, n'importe quoi !!! donc ne faites pas ensuite de tests du type : `if (joueur_1_a_joue==NULL),` ça a le dont d'énerver furieusement l'ordinateur…

ANNEXE B. Commenter ses programmes

Vous savez depuis la P1 qu'il est important de commenter vos programmes. Il ne faut pas trop en mettre non plus; la règle étant que ces derniers doivent être pertinents.

Il est conseillé de commenter ses programmes au fur et à mesure.

Exemple d'entête de fonction :

```
//*****************************************************************************************
//* Fonction          : TfrmProgPrincipalConnexionBase ( )                                *
//*                                                                                       *
//* Objet             : Connexion à la base de données MySQL.                             *
//*                                                                                       *
//* Paramètres        : Aucun.                                                            *
//*                                                                                       *
//* Renvoie           : Booléen, qui indique si la connexion a bien eu lieu.              *
//*                                                                                       *
//* Commentaires      : Pour se connecter à la base, on utilise l'API MySQL.              *
//*                     La base s'appelle GEODE, toutes les tables sont contenues         *
//*                     dans cette base.                                                  *
//*                                                                                       *
//* Historique                                                                            *
//*                                                                                       *
//* Date              Commentaires                                  Nom                   *
//* ========          ====================================      =========                 *
//* 02/06/05          Création.                                     Bruno GOYET           *
//*                                                                                       *
//*****************************************************************************************
int TfrmProgPrincipalConnexionBase( )
{

…
}
```

ANNEXE C. Qu'est-ce qu'un éditeur de liens ?

On part d'un fichier source...

- Outil : Editeur de textes.

```
#include <stdio.h>
main(){
          float moy, somme, valeur;
          int i;
         printf ("valeur ?\n");
          .......
}
```

... On en tire un fichier objet...

fichier objet de nom : *moyenne.o*
01100100010001110100111001010000100 01000100100011110101100110101010011 1 00001001010111111101001010100100 1010

Les fonctions printf, scanf, feof ne sont pas présentes dans le fichier objet. Le compilateur se contente de traduire l'appel de ces fonctions du langage C vers le langage machine.

- Outil : Compilateur.

Unix
`gcc -c moyenne.c`

Sous Unix (Linux), le compilateur C se lance avec la commande gcc.

L'option `-c` qui figure en complément de la commande `gcc` définit la nature du traitement à effectuer : la compilation du fichier indiqué à la suite (`moyenne.c`)

Le compilateur, à partir du fichier `moyenne.c`, produit un fichier `moyenne.o`

...puis un fichier exécutable

nom du fichier exécutable : *moyenne*
01100100010001110100111001010000100 01000100100011110101100110101010011 1 00001001010111111101001010100100 1010 11100101001000100111101000010010001000100010 01001001000000000100010000000100100100100010 10101010101010010000100100000001001001001001

Les fonctions printf, scanf, feof cette fois sont présentes dans le fichier objet. Car, précisément, cela fait partie du travail de l'éditeur de liens d'aller chercher, dans les différentes bibliothèques système, les fonctions nécessaires à l'exécution de l'application, et de les incorporer au fichier exécutable.

- Outil : Editeur de liens.

Unix
`gcc -o moyenne moyenne.o`

Sous Unix (Linux), l'éditeur de liens, comme le compilateur se lance avec la commande gcc.

L'option `-o` (o comme output) qui figure en complément de la commande `gcc` définit la nature du traitement à effectuer : la création d'un fichier exécutable dont le nom est fourni juste après (`moyenne`).

Cette création se fait à partir de l'édition des liens des fichiers objets donnés en arguments (ici, il n'y en a qu'un, `moyenne.o`).

L'éditeur de liens, à partir du fichier `moyenne.o`, produit donc le fichier exécutable `moyenne`

La commande gcc est en fait une commande générale qui lance le compilateur (`/usr/ccs/bin/cc`) ou l'éditeur de liens (`/usr/ccs/bin/ld`) en fonction des arguments et des options.

On peut aussi obtenir un exécutable à partir du fichier source en une seule étape.

- Outil : Compilateur - Editeur de liens.

Unix
`gcc -o moyenne moyenne.c`

La commande gcc ci-dessus produit l'exécutable `moyenne` à partir du fichier source `moyenne.c`

Dans ce cas, elle commence par lancer le compilateur sur le fichier `moyenne.c`, puis elle lance l'éditeur de liens sur le fichier temporaire `moyenne.o` obtenu à l'issu de la compilation, produit l'exécutable `moyenne`, puis détruit le fichier temporaire `moyenne.o`

ANNEXE D. Quelques actions utiles

D.1 Firefox ne démarre pas car il serait déjà démarré

- Ouvrir un terminal
- aller dans le répertoire home de l'étudiant

  `>cd`
- afficher l'ensemble des dossiers et fichiers cachés ou non du répertoire

  `>ls –la`
- vérifier que le dossier « .mozilla » est présent
- si oui,

  `>rm –rf .mozilla`
- redémarrer firefox