Université Gustave Eiffel — INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

L2 Info - 2022-2023
Programmation C
Projet 1° partie

Enveloppe Convexe dans le plan

L'objectif est de vous faire implémenter un algorithme de détermination d'enveloppe convexe dans le plan en utilisant une structure de type `liste doublement chainee bouclée`.

Ce sujet sera suivi (très bientôt) par une seconde partie, suite directe de celle-ci.

Préambule - Un tout petit peu de math...

Principe général

On part d'un ensemble $\mathcal{E}_n$ de n points de $\mathbb{R}^2$, distribués dans le plan (peu importe comment pour l'instant). L'enveloppe convexe, $EC_{\mathcal{E}_n}$, de cet ensemble de points est le plus petit (en surface) polygône convexe englobant tous les points. Tous les sommets de $EC_{\mathcal{E}_n}$ sont des points de $\mathcal{E}_n$, et tout point de $\mathcal{E}$ est soit à l'intérieur de $EC_{\mathcal{E}_n}$, soit un sommet de $EC_{\mathcal{E}_n}$.

Il existe de nombreux algorithmes[(1)] pour déterminer ce polygône. En voici un de plus.

ⓐ Initialisation : pour un ensemble $\mathcal{E}_3$ à trois points, l'enveloppe $EC_{\mathcal{E}_3}$ est nécessairement le triangle formé par ces trois points.

ⓑ Cas général : supposons que l'on ait déterminé l'enveloppe $EC_{\mathcal{E}_p}$ pour un ensemble à $(p \geq 3)$ points. Ajoutons un $(p+1)^{eme}$ point :

- soit il se trouve à l'intérieur du polygône précédent, alors $EC_{\mathcal{E}_{p+1}} = EC_{\mathcal{E}_p}$ (il n'y a donc rien à faire)
- soit il est à l'extérieur et il faut construire un nouveau polygone $EC_{\mathcal{E}_{p+1}}$, englobant $EC_{\mathcal{E}_p}$ et dont un des sommets est ce dernier point.

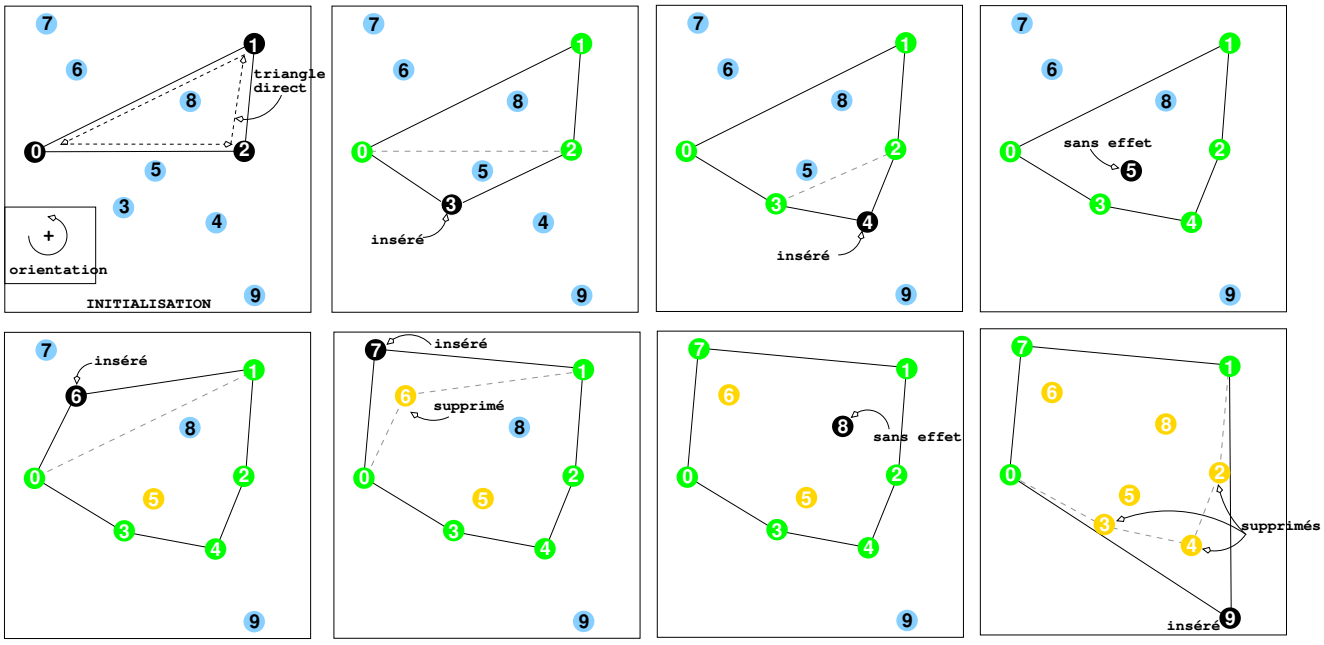


Figure 1: *Etapes de la construction de l'enveloppe d'un ensemble à 10 points*

(1) *cf.* https://fr.wikipedia.org/wiki/Enveloppe_convexe

Comme le montre l'exemple de la Fig.1 (pour une ensemble à 10 points), cette insertion d'un nouveau sommet modifie localement le polygone : soit le point s'ajoute simplement à la liste des sommets (cf. Fig.1 cadres 2, 3, 5), soit il remplace un ou plusieurs autres sommets consécutifs (cf. Fig.1, cadres 6, 8). Ainsi, pour chaque nouveau point traité, le nombre de points sur l'enveloppe (sa `longueur`) augmentera au plus de 1 mais pourra diminuer de manière considérable.

Le coeur de la méthode consiste donc à déterminer la position d'un point par rapport aux segments constituant les arêtes du polygône.

Orientation et positionnement

Pour que cette méthode fonctionne il est primordial de considérer que tout, dans le plan, est *orienté*. Cette orientation est purement arbitraire (ont choisit en général le sens trigonométrique[(2)]) mais conditionne toute la suite.

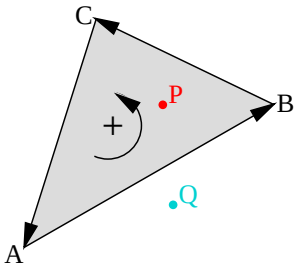
En conséquence, pour 3 points distincts A B et C du plan, les triangles $(A \to B \to C)$ et $(A \to C \to B)$ ne sont pas du tout les mêmes. L'un à forcément une orientation positive et l'autre une orientation négative, et ce, même si les 3 points sont alignés.

Cela induit une orientation des arêtes du triangle (ce sont des *vecteurs*) : $\left(\overrightarrow{AB}, \overrightarrow{BC}, \overrightarrow{CA}\right)$ ou $\left(\overrightarrow{AC}, \overrightarrow{CB}, \overrightarrow{BA}\right)$.

Ainsi tout vecteur $\overrightarrow{AB}$ définit un demi-plan à gauche et un demi-plan à droite et un triangle orienté (A, B, C) définit un intérieur et un extérieur : un 4° point P sera à l'*intérieur* du triangle si et seulement si il est du même côté pour chaque arête.

Si le plan est orienté dans le sens *trigo+*, l'intérieur du triangle est l'espace situé *à gauche de chacune des 3 arrêtes.*

Dans l'exemple ci-contre, le point P est à gauche des 3 arêtes : il est à l'intérieur du triangle. Le point Q est est à gauche de $\overrightarrow{BC}$ et $\overrightarrow{CA}$, mais à droite de $\overrightarrow{AB}$: il est donc à l'extérieur du triangle *par rapport à l'arête* (AB)
☞ *il sera inséré dans l'enveloppe entre les sommets* A *et* B.



Ce qui est vrai pour un triangle se généralise très simplement à un polygône quelconque, et en particulier à un polygône convexe. C'est cette proprité qui va nous permettre de positionner très simplement un point par rapport à un polygône.

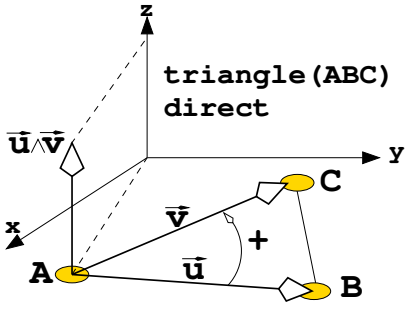
Orientation d'un triangle

Pour déterminer une orientation dans le plan on utilisera, par exemple, l'opérateur *Produit Vectoriel* [(3)], définit (dans $\mathbb{R}^3$) par :

$$\vec{u}\begin{vmatrix} x_u \\ y_u \\ 0 \end{vmatrix} \wedge \vec{v}\begin{vmatrix} x_v \\ y_v \\ 0 \end{vmatrix} = \vec{w}\begin{vmatrix} 0 \\ 0 \\ x_u * y_v - y_u * x_v \end{vmatrix}$$

Dans $\mathbb{R}^2$, on peut l'écrire plus simplement, sous forme scalaire :
$\vec{u} \wedge \vec{v} = (x_u * y_v - y_u * x_v)$
Ainsi, un triangle (A, B, C) sera dit *direct* (positif) si il vérifie $\overrightarrow{AB} \wedge \overrightarrow{AC} \geq 0$ soit $(x_B - x_A) * (y_C - y_A) - (x_C - x_A) * (y_B - y_A) \geq 0$

Le calcul de positionnement d'un point par rapport à une arête d'un polygône (un segment orienté, c'est-à-dire un vecteur) est donc très simple et économique : 5 additions, 2 multiplications.



Point de vigilance : le cas où les 3 points (A, B, C) sont alignés ($\overrightarrow{AB} \wedge \overrightarrow{AC} = 0$) est un cas limite. On considèrera ici qu'un tel triangle est *positif.*

☞ un point P situé exactement sur une arête du polygône sera considéré comme étant *à l'intérieur.*

---

(2) *sens contraire des aiguilles d'une montre souvent appelé* trigo+ *(ou* counterclockwise *en anglais)*

(3) cf. https://fr.wikipedia.org/wiki/Produit_vectoriel

Retour sur l'algorithme d'Enveloppe Convexe

Avec cette méthode de positionnenement on peut maintenant détailler un peu plus l'algorithme itératif de construction de l'enveloppe convexe d'un ensemble de points.

ⓐ Initialisation

On initialise l'enveloppe par un triangle formé des 3 premiers points P_0, P_1 et P_2 de l'ensemble, ordonnés de telle sorte que le triangle soit direct. L'enveloppe initiale sera donc (P_0, P_1, P_2) ou (P_0, P_2, P_1) selon le signe de $\overrightarrow{P_0P_1} \wedge \overrightarrow{P_0P_2}$ (cf. Fig.1, cadre 1).

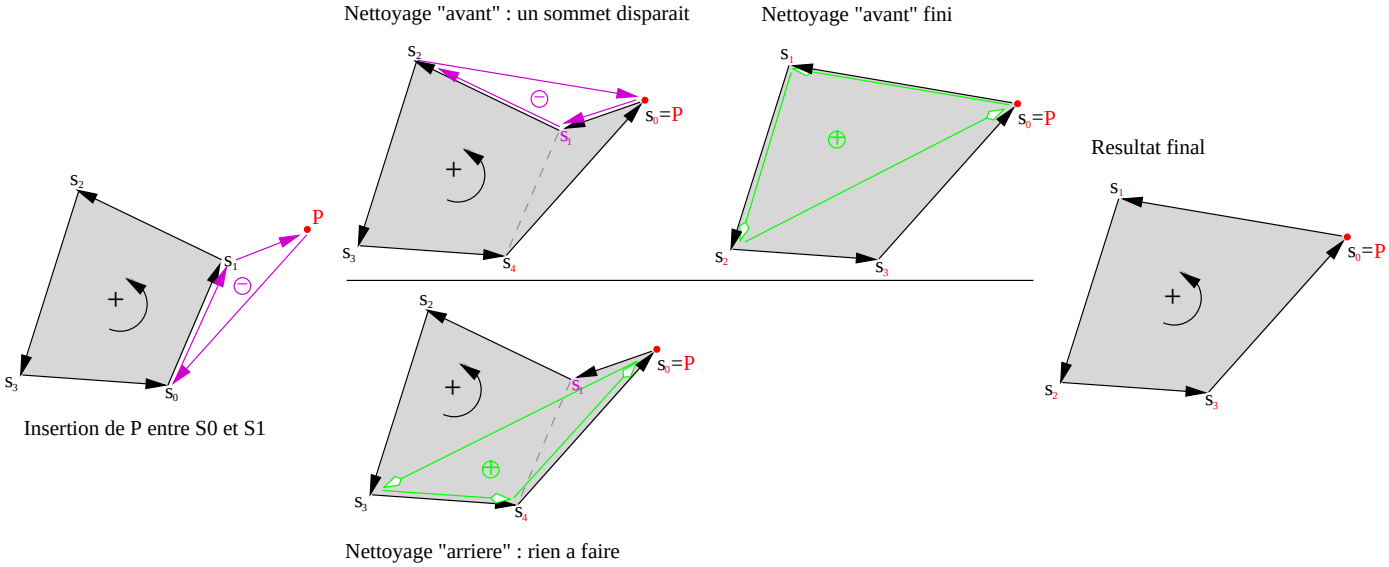
ⓑ Traitement d'un point

Pour déterminer si un point P est à l'intérieur ou à l'extérieur d'un polygône convexe, il faut parcourir les arêtes $\overrightarrow{S_i, S_{i+1}}$ (*orientées*) de ce polygône et tester si le triangle ainsi formé (P, S_i, S_{i+1}) est direct ou non (même méthode que pour l'initialisation):

- Si il est **direct**, on ne peut rien conclure et il faut tester le triangle suivant formé des points (P, S_{i+1}, S_{i+2}).

  ☞ Si, pour toutes les arêtes du polygone convexe courant, les triangles (P, S_i, S_{i+1}) sont direct, celà signifie que le point P est à l'intérieur. On ne fait rien et on passe au traitement du point suivant de l'ensemble.

- Si il est **indirect**, cela signifie que P est à l'*extérieur* de l'enveloppe, par rapport au segment $[S_i, S_{i+1}]$. Il faut alors :
  - insérer P dans la liste enveloppe entre S_i et S_{i+1}. Pour plus de simplicité, on considère que ce nouveau sommet (P) devient le point d'entée du polygône (S_0).
  - après cette opération, certains sommets de l'ancienne enveloppe, avant et/ou après P, peuvent être passés à l'*intérieur* de la nouvelle. Il sera alors nécessaire de les supprimer.
    Pour celà il suffit de reparcourrir l'enveloppe dans les deux sens[4] :
    - nettoyage "avant" : à partir de $P = S_0$ on teste à nouveau le triangle (S_0, S_1, S_2) grâce à la méthode de position. Si il est **in**direct, cela signifie que le sommet pointé par S_1 est passé à l'intérieur; on le supprime, et on teste le suivant. Sinon on s'arrête.
    - nettoyage "arrière" : on fait la même chose dans l'autre sens, à partir du précédent. Si (S_0, S_{-2}, S_{-1}) (attention à l'ordre !) est **in**direct, on supprime S_{-1}, sinon on s'arrête.



[4] *l'ordre dans lequel se font ces 2 "nettoyages" n'a normalement aucune importance.*

La mise en oeuvre

Ce projet est assez conséquent et présente différentes étapes et différents prolongements plus ou moins indépendant. Il est **fortement** conseillé de procéder par étape, en réalisant et conservant **plusieurs versions**. D'une version à l'autre même les structures de données sont susceptibles d'évoluer (gestion des couleurs, des fichiers...).
☞ Lorsqu'une étape est correcte, **sauvegardez-là** dans une archive spécifique et passez à l'étape suivante à partir d'une **copie** de la version qui fonctionne.
☞ Il vaut mieux rendre 1 seule étape qui fonctionne que 2 ou 3 qui ne fonctionnent pas

Structure de données

Point, Vertex, Polygône, Enveloppe

La nature même de cette méthode induit la structure de données que nous allons utiliser :

ⓐ On travaille sur des polygônes, formés de sommets consécutifs, donc d'arêtes orientées et nous allons ajouter et/ou retirer des sommets[(5)] à ces polygônes. La structure de donnée la plus adaptée est donc la **liste chaînée**.

ⓑ Ces polygônes étant fermés, une liste **bouclée** (le suivant du dernier sommet est le premier) s'impose.

ⓒ A chaque ajout de sommet, l'algorithme étant susceptible de supprimer des sommets *avant* et *après* le point inséré, un **double chaînage** (avant/arrière) sera beaucoup plus simple à manipuler.

ⓓ L'évolution du nombre de point sur l'Enveloppe Convexe (Convex Hull, en anglais) est une donnée intéressante à suivre pour évaluer la complexité de la méthode. Ces informations (longueurs courante, moyenne, maximale...) seront intégrées à la structure de données pour traitement ultérieur.

☞ Un Polygône sera représenté par une **liste doublement chaînée bouclée** de Vertex
☞ Une Enveloppe Convexe sera définie par un Polygône et deux int pour suivre l'évolution de sa longueur

```
typedef struct _vrtx_
{
  Point*    s;         /* un point de l'ensemble */
  struct _vrtx_ *prev; /* le vertex précédent    */
  struct _vrtx_ *next; /* le vertex suivant      */
} Vertex, *Polygon;
```

```
typedef struct
{
  Polygon pol;    /* le polygône          */
  int     curlen; /* la longueur courante */
  int     maxlen; /* la longueur maximale */
} ConvexHull;
```

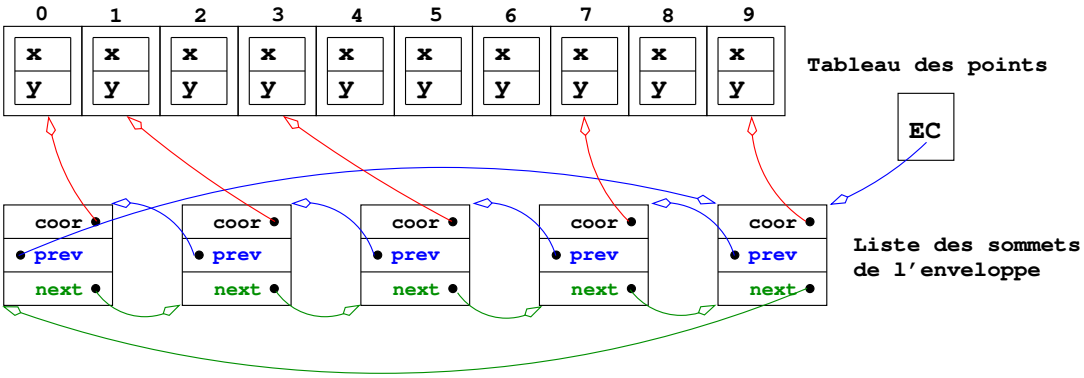


Figure 2: *Structure de données : Etat final de l'exemple de la Fig.1*

☞ A ces différents objets géométrique pourront (devront !) être associées des couleurs, des formes... A vous de gérer cela au mieux en fonction des options – cf. la suite.
La "qualité" (pertinence, esthétique) graphique compte dans ce projet.

[(5)] *appelé Vertex : point d'intersection entre deux ou plusieurs segments dans une construction géométrique*

Etape 1 : génération du nuage de Points (l'ensemble $\mathcal{E}_n$)

☞ Point de vigilance : cf. cadre en toute fin de sujet

Votre programme devra proposer au moins 2 façons de distribuer les points dans la fenêtre :

① A la souris : un clic dans la fenêtre ajoute un point à l'ensemble et effectue son traitement par rapport à l'enveloppe.

② Génération du nuage de point par distribution aléatoire *contrôlée* :

- ⓐ dans un carré de centre $C(x_c, y_c)$ et de demi-côté r (l'enveloppe convexe tendra vers ce carré)
- ⓑ dans un cercle de centre $C(x_c, y_c)$ et de demi-côté r (l'enveloppe convexe tendra vers ce cercle)
- ⓒ pour chacune de ces formes, une distribution en pseudo-spirale : le rayon r de la forme augmente au fur et à mesure que les points arrivent.
  ☞ l'enveloppe, d'abord concentrée autour du centre, va grandir et converger vers la forme choisie (carré ou cercle).

Etape 2 : traitement des points

Quel que soit le mode de génération, les points seront traités séquentiellement, c'est-à-dire dans leur ordre d'apparition mais votre programme devra proposer 2 modes de traitement/visualisation

① Affichage dynamique : on actualise l'affichage (points et enveloppe) à chaque nouveau point traité
☞ cela permet de visualiser l'évolution du processus, mais le ralenti (l'affichage peut devenir coûteux)

② Affichage terminal : on tire tous les points, on lance le calcul de l'enveloppe et on n'affiche que le résultat final.
☞ cela permet de mieux évaluer les performances de l'algorithme lui-même en évitant les coûts d'affichage lorsque le nombre de point devient important.

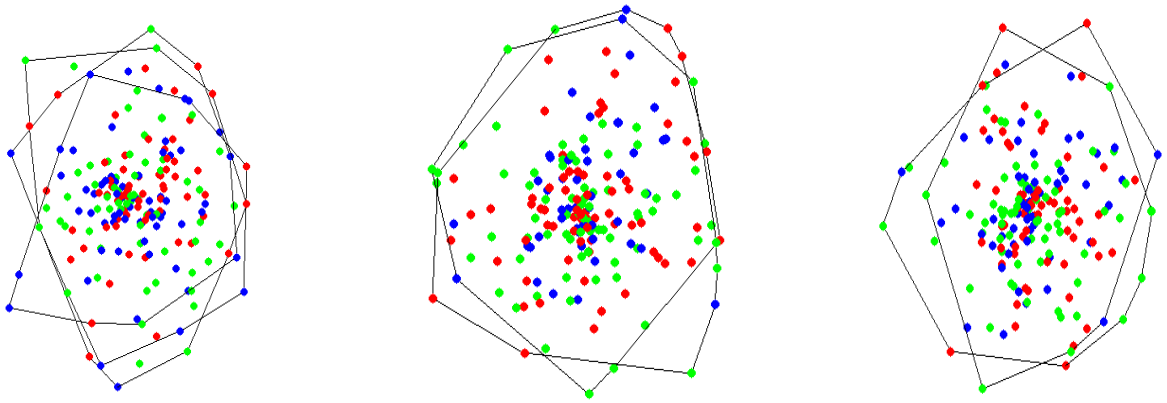
Consignes diverses

Environnement graphique interactif

- La partie graphique de ce projet est bien sûr à réaliser avec la LibMLV.
- L'application pourra s'ouvrir sur un menu permettant de selectionner les options : mode de distribution des points (à la souris, aléatoire contrôlé, forme), mode de déroulement / affichage (point par point, dynamique, terminal)....
- L'affichage (dynamique ou terminal) de quelques informations statistiques (longueurs courante, moyenne, maximale) de l'enveloppe) sera le bienvenu
- Des options permettant par exemple de quitter l'application à tout moment, de la relancer automatiquement... seront également appréciées.

☞ De la créativité et de la rigueur pour produire une application performante et conviviale.

ATTENTION :
La libMLV (cf. les TP que vous avez déjà réalisé) travaille essentiellement avec des entiers. Un point est donc généralement définit avec deux coordonnées *entières*. Bien que ce ne soit pas gênant a priori, cela peut provoquer des effets indésirables lorsque 2 points se trouvent confondus : le calcul de positionnement/orientation peut produire une valeur erronnée et déclencher un dysfonctionnement de l'algorithme, comme dans les exemples ci-dessous.



☞ Il est donc préférable ici de travailler avec des coordonnées réelles (double). Cela diminue drastiquement les risques.

☞ Pour les points définis au "clic-souris" (via la fonction void MLV_wait_mouse(int *x, int *y);) une petite perturbation aléatoire des valeurs permet de régler ce problème.

Quelque chose dans ce goût-là (vous pouvez certainement faire mieux) :

```
Point get_point_on_clic()
{
  double PERTURB = 0.0001/RAND_MAX;
  Point P;
  int x,y;
  MLV_wait_mouse(&x,&y);
  P.x = x+(rand()%2?+1.:-1.)*PERTUB*rand();
  P.y = y+(rand()%2?+1.:-1.)*PERTUB*rand();
  return P;
}
```

Rendre le projet

Votre projet devra être rendu sous la forme d'une d'archive `L2S3.Projet.NOM1.NOM2.[tgz/zip]` à déposer sur eLearning.

Le projet devra être réalisé seul ou en binôme (donc au plus deux personnes....). Les deux personnes participent à part égale au projet. Si le correcteur a un doute sur la compétence d'un élément du binôme sur le projet, il pourra mettre deux notes distinctes. Si un projet est rendu `n` fois (triche, plagiat) sa note[(6)] sera divisée par `n`, y compris pour la "version originale".

Votre projet devra être bien documenté (commenter les fonctions, expliciter les variables, sans en faire trop...).

D'autre part, il devra être accompagné d'un fichier texte `README` précisant clairement vos noms, prénoms, groupes de TD/TP et expliquant le fonctionnement de votre projet : comment le compiler, où trouver le fichier exécutable, quelles sont les options de la ligne de commande (quelques exemples d'utilisation).... bref tout ce qui peut faciliter l'usage de votre projet et mettre le correcteur dans de bonnes dispositions... Ce fichier précisera aussi le plus clairement possible ce que vous **n'**avez **pas** réussi à faire ou ce qui ne marche pas comme vous voudriez.

Un rapport rédigé (au format `PDF`) sera le bienvenu. Il n'est pas obligatoire mais rapportera un bonus si votre code est un peu faible

De même une documentation du code grâce à l'outil `doxygen` plutôt que de simples commentaire sera appréciée à sa juste valeur.

Important
Vos correcteurs reçoivent beaucoup de projets : si vous voulez être notés équitablement respectez les règles suivantes :

- Au moment de l'archivage votre répertoire ne doit rien contenir d'autre que vos codes sources utiles et le fichier `README` mais ne doit contenir ni fichiers objets, ni exécutable... purgez les versions de code inutiles... bref, rendez une archive propre.
- Votre programme doit *impérativement* fonctionner (compilation/exécution) sur les machines `linux` de l'université (salle de TP). L'argument classique *"Oui, mais chez moi ça marche bien.."* ou *"Je l'ai fait sur mon Mac"* est irrecevable.
- N'envoyez qu'*une seule* archive – si possible la bonne...
- Ne prenez pas les correcteurs pour des imbéciles : ils savent très bien détecter les projets *pompés* et disposent de détecteurs de plagiat.

Ces conseils sont évidemment valables pour les autres projets que vous aurez à faire cette année et les suivantes...

Options

Les options, variantes et améliorations possibles sont nombreuses. Certaines vous seront demandées sous peu dans une seconde partie.

Vous avez carte blanche psour les options mais n'implantez pas d'option tant que le reste n'est pas parfaitement fini. Mieux vaut un projet sans option bien propre et qui fonctionne qu'un projet avec beaucoup d'options qui est mal construit ou qui ne fonctionne pas.

[(6)] *sans préjudice de sanctions complémentaires!*