


STS 1 SN IR 2021-2022	PROGRAMMATION STRUCTUREE DS – 4H00	
---------------------------------	--	---

TRAVAIL DEMANDE - RECOMMANDATIONS

UN LECTURE COMPLETE ATTENTIVE DE L'INTEGRALITE DU SUJET EST RECOMMANDEE AVANT D'ENTAMER TOUT DEVELOPPEMENT OU CODAGE.

Une base applicative est fournie sous l'archive « NomPrenom_Base.zip » contenant un projet configuré Eclipse/SDL/GCC.

Après importation sous Eclipse de cette base, renommer aussitôt le projet, comme à l'accoutumée, avec vos <NomPrenom>. Renommer ensuite le fichier principal contenant la fonction main() avec également vos <NomPrenom.c>.

Dans les entêtes des fichiers .h et .c, indiquer vos nom et prénom en lieu et place prévu à cet usage.

Le travail demandé consiste à l'élaboration des codes répondant aux fonctionnalités attendues aux emplacements marqués « ToDo ». **VOUS N'EFFACEREZ PAS CES MARQUES MAIS PLACEREZ LE CODE DEMANDE EN DESSOUS DE CHACUNE D'ELLES.**

Il est également recommandé de procéder par étapes constructives dans l'élaboration de l'application, on la construisant pas à pas, et en testant chaque étape avant d'empiler la suivante.

A REMETTRE - RECOMMANDATIONS

Le projet complété et paramétré avec vos nom et prénom, sera archivé au format .zip par l'utilitaire d'exportation intégré à Eclipse. L'archive devra elle-même être nommée de vos nom et prénom, qui on le rappelle, SANS ACCENTS SANS ESPACES PREMIERE LETTRE DES MOTS EN MAJUSCULE.

Cette archive sera à déposer sous Moodle.

CONCEPTION LOGICIELLE :

« Stars »

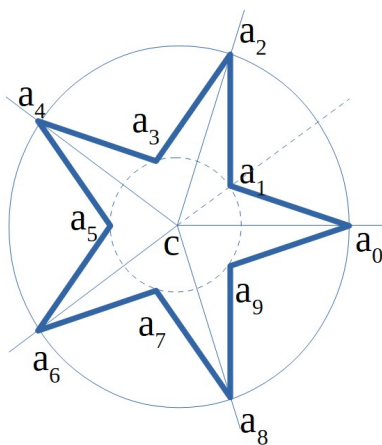
1 DESCRIPTION

On conçoit une application qui anime des objets graphiques mobiles dessinés sous formes d'étoiles avec un nombre de branches, de tailles et de couleurs variables. Ces étoiles, générées aléatoirement en nombre, en position et en vitesse de déplacement, se meuvent dans une zone prédéfinie, et sont détruites une fois que celles-ci franchissent les limites de cette zone.

Du point de vue structurel, un objet étoile est défini par son nombre de branches, sa taille qui sera caractérisée par le rayon du disque l'englobant, sa position initiale lors de sa création, sa vitesse de déplacement et sa couleur.

Du point de vue graphique, une étoile est un polygone défini par un nombre n de sommets reliés deux à deux par une ligne, pour former un contour fermé. On distingue ainsi pour une étoile, les sommets de rangs pairs (a_0, a_2, a_4, \dots) situés sur le disque de rayon r englobant la figure, et les sommets de rangs impairs (a_1, a_3, a_5, \dots) situés sur un disque interne dont le rayon sera une fraction du rayon r du disque englobant.

La position angulaire de chaque sommet, est décalée l'une par rapport à l'autre d'un angle valant $2\pi/n$, où n est le nombre de sommets du polygone. Si le premier sommet a_0 est positionné avec un angle nul (origine des angles), le sommet a_1 aura un angle de $2\pi/n$, le sommet a_2 un angle de $2 \times 2\pi/n$, ... et de manière générale, le sommet a_k de rang k un angle égal à $k \times 2\pi/n$.



Une étoile à 5 branches, formée d'un polygone de 10 sommets notés a_0 à a_9 .

Les sommets seront repérés par rapport au centre c du polygone.

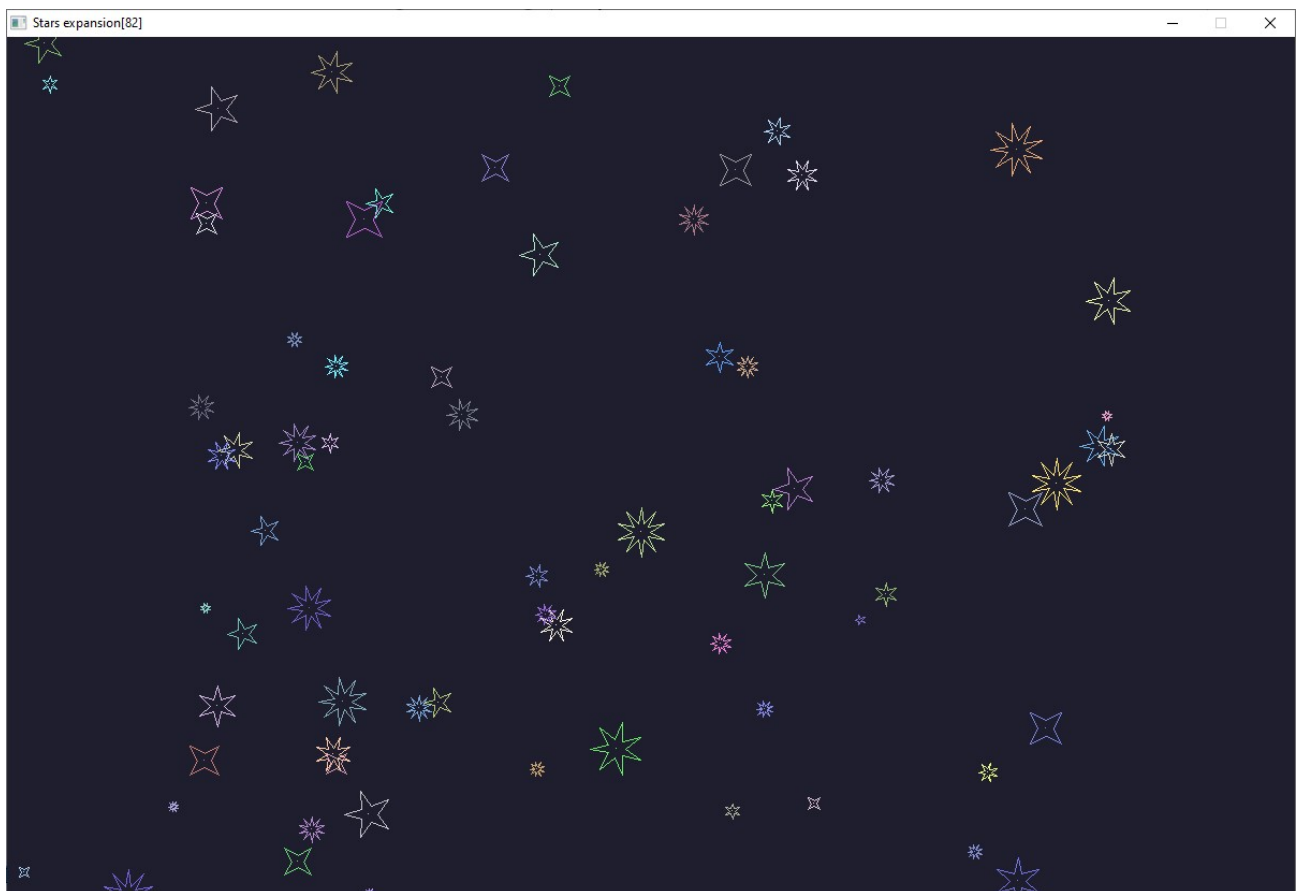
Dans ces conditions les coordonnées cartésiennes **(x, y)** de chaque sommet se calculent à l'aide des expressions données ci-dessous :

$$a_k \begin{cases} x_k = x_c + r \cdot \cos(k \cdot 2\pi/n) \\ y_k = y_c + r \cdot \sin(k \cdot 2\pi/n) \end{cases}$$

Avec **x_k y_k** les coordonnées du sommet **k**, **x_c y_c** les coordonnées du centre **c** du polygone, **k** le rang du sommet considéré, **n** le nombre total de sommets du polygone et **r** le rayon du disque sur lequel est positionné le sommet **k**. Attention, la valeur **r** dépend de la parité du rang du sommet considéré.

Ainsi défini, le centre **c** du polygone est le barycentre des sommets, et ses coordonnées se déduisent de celles des sommets à l'aide des expressions données ci-dessous :

$$c \begin{cases} x_c = (x_0 + x_1 + x_2 + \dots + x_{n-1}) / n \\ y_c = (y_0 + y_1 + y_2 + \dots + y_{n-1}) / n \end{cases}$$



Un aperçu d'une série d'étoiles générées de manière aléatoire.

2 MODELISATION LOGICIELLE

2.1 Entité « Star » :

2.1.1 Structure :

Une entité « star » sera définie par la structure donnée ci-dessous.

```
struct s_star{
    SDL_Point    m_ptCenter;        /* Star center coordinates */
    SDL_Point    m_ptVelocity;      /* Star speed vector */
    SDL_Color    m_color;           /* Star RGBA color */
    size_t       m_szBranches;      /* Star branches number */
    int          m_iRadius;         /* Star global radius */
    SDL_Point*   m_pVertices;       /* Pointer to the star polygon vertices */
};
```

Vertex du polygone définissant le tracé de l'étoile :

vertex = sommet d'une forme géométrique, également vertice en anglais. Un vertex désigne ici les coordonnées cartésiennes (x, y) d'un sommet du polygone formant l'étoile.

Le nombre de sommets d'une étoile est lié à son nombre de branches dans un rapport double :
nombre de sommets = 2 × nombre de branches.

La fonction de la SDL pour le tracé du polygone est donnée ci-dessous :

```
/**
 * \brief Draw a series of connected lines on the current rendering target.
 *
 * \param renderer The renderer which should draw multiple lines.
 * \param points The points along the lines
 * \param count The number of points, drawing count-1 lines
 *
 * \return 0 on success, or -1 on error
 */
int SDL_RenderDrawLines(SDL_Renderer * renderer,
                        const SDL_Point * points,
                        int count);
```

Cette fonction trace les droites entre les points désignés par le paramètre « **points** », un pointeur sur un ensemble (tableau) d'éléments de type **SDL_Point**, contenant « **count** » éléments. **Pour obtenir un polygone fermé, le dernier point doit être identique au premier point.**

Ainsi par exemple, si l'on désire tracer un polygone à 5 côtés, il faudra définir une structure (tableau) contenant 5 + 1 points de type **SDL_Point**, dont le premier et le dernier point auront les mêmes valeurs en x et en y.

2.1.2 Fonctionnalités :

```
t_star*StarNew(SDL_Point ptCenter, SDL_Point ptVelocity, size_t szBranches, int iRadius,
               SDL_Color color);
```

Réserve dynamiquement l'espace mémoire nécessaire à la structure **t_star**, initialise les champs de celle-ci avec les paramètres passés à la fonction et retourne le pointeur sur la structure ainsi créée et initialisée.

Vertex du polygone :

Un tableau dynamique comportant le **nombre de sommets + 1** éléments de type **SDL_Point** sera réservé dynamiquement. Chaque vertex (élément de type **SDL_Point**) sera ensuite initialisé à l'aide des expressions en **page 2** donnant x_k et y_k (vertex a_k). Le tableau comporte un vertex de plus que le nombre de sommets du polygone, ce dernier ayant les mêmes coordonnées que le premier vertex du tableau pour permettre le tracé d'un polygone fermé avec la fonction **SDL_RenderDrawLines()**.

L'utilisation des fonctions trigonométriques $\sin()$ et $\cos()$ nécessite l'inclusion de la bibliothèque `<math.h>` dans les fichiers concernés, ainsi que le paramétrage du projet avec l'option `-lm` pour le linker. CES PARAMETRAGES SONT DEJA POSITIONNES DANS L'APPLICATION DE BASE.

Les fonctions trigonométriques sinus et cosinus utilisent des angles exprimés en radians.

La constante π est définie dans la bibliothèque `<math.h>` à l'aide de la constante symbolique `M_PI`.

Les vertex de rangs pairs sont placés sur le rayon de l'entité « star ». Les vertex de rangs impairs sont placés sur un rayon égal à une fraction du rayon de l'entité. Cette fraction est paramétrée à l'aide de la constante symbolique **STAR_INTERNAL_RADIUS_RATIO** situé dans le fichier "star.h".

```
t_star*StarDel(t_star*pStar);
```

Restitue les ressources mémoire initialement requises par l'entité star.

```
t_star*StarDraw(const t_star*pStar, SDL_Renderer*pRenderer);
```

Réalise le tracé du polygone représentatif de l'entité star.

On pourra également matérialiser le centre de l'étoile à l'aide de la fonction de la SDL donnée ci-dessous :

```
int SDL_RenderDrawPoint(SDL_Renderer * renderer, int x, int y);
```

```
t_star*StarMove(t_star*pStar, SDL_Rect*pSpace);
```

Réalise le déplacement de l'entité star en lui appliquant son vecteur vitesse. Chaque vertex, ainsi que le centre de l'entité se voient donc ainsi augmentés des composantes du vecteur vitesse.

```
t_star*StarOffset(t_star*pStar, SDL_Point*pOffset);
```

Déplace l'entité star de la quantité contenue dans le paramètre **pOffset**. Chaque vertex, ainsi que le centre de l'entité se voient donc ainsi augmentés des composantes du paramètres **pOffset**.

2.2 Entité « App » :

2.2.1 Structure :

L'entité « app » sera définie par la structure donnée ci-dessous.

```
struct s_app{
    uint32_t      m_uStatus;           /* application status flags */
    SDL_Window *  m_pWindow;           /* main window structure */
    SDL_Renderer* m_pRenderer;         /* main renderer structure */
    SDL_TimerID   m_timerID;           /* animation timer ID */

    /*****

    SDL_Rect      m_space;             /* the space area delimiter */
    t_container * m_pStars;            /* the stars container */
    int           m_generateTimer;     /* remaining time before star re-generating */
    SDL_Point     m_generateOrigin;    /* re-generation area location */
    SDL_Point     m_ptStart;           /* for mouse location tracking */
};
```

```
uint32_t      m_uStatus;           /* application status flags */
```

Maintient les flags d'état de l'application.

```
enum s_statusMasks{
    ST_AL_CLEARED           = 0x00000000,
    ST_APP_INIT_FAILED     = 0x80000000,
    ST_APP_SDL_INITIATED   = 0x00000001,
    ST_APP_MOUSE_BUTTON_DOWN = 0x00000002,
    ST_APP_PAUSED          = 0x00000004,
};
```

ST_APP_MOUSE_BUTTON_DOWN Indique si Le bouton de La souris est enfoncé ou non

ST_APP_PAUSED Indique si L'animation est en mode pause ou non

```
SDL_Rect      m_space;           /* the space area delimiter */
```

Définit la zone dans laquelle les objets graphiques évoluent « librement ». En dehors de cette zone, les objets sont détruits. Cette zone est paramétrée avec les constantes suivantes :

- APP_WINDOW_WIDTH
- APP_WINDOW_HEIGHT
- APP_SPACE_PADDING

```
t_container *m_pStars;           /* the stars container */
```

Contient et maintient les entités « stars ».

```
int      m_generateTimer;           /* remaining time before star re-generating */
```

Compteur indiquant le temps restant exprimé en unités d'animation avant la ré-génération d'une nouvelle série d'étoiles. La valeur initiale de ce compteur est tirée de manière aléatoire à

chaque fois que celui-ci atteint la valeur 0. La valeur aléatoire est paramétrée par les constantes suivantes :

- `APP_STAR_TIMER_GEN_MAX`
- `APP_STAR_TIMER_GEN_MIN`

```
SDL_Point  m_generateOrigin;  /* re-generation area location */
```

Définit le point autour duquel la zone de régénération des entités « stars » se produit. Ce point suivra le mouvement du pointeur de la souris. Les dimensions de la zone de génération, de forme carrée, sont paramétrées par la constante :

- `APP_STAR_AREA_GEN_SIZE`

```
SDL_Point  m_ptStart;          /* for mouse location tracking */
```

Champ nécessaire au suivi du mouvement du pointeur de la souris.

2.2.2 Fonctionnalités :

```
t_app*AppNew(void);
```

Réserve dynamiquement l'espace mémoire nécessaire à la structure **t_app**, initialise les champs de celle-ci et crée les diverses entités nécessaires à l'application.

```
t_app*AppDel(t_app*pApp);
```

Restitue les ressources allouées dynamiquement lors de la phase de « construction » de l'application.

```
int AppRun(t_app*pApp);
```

Implémente la partie active de l'application en procédant au traitement des événements s'y rattachant.

```
Uint32 _AppTimerCallback(Uint32 interval, t_app*pApp);
```

Implémente la partie animation de l'application. Prend en charge le dessin et le mouvement des entités graphiques.

3 DEVELOPPEMENT

3.1 Entité « Star »

3.1.1 Fonctionnalités de base

➤ **Implémenter dans un premier temps les fonctions suivantes du module « star » :**

- `StarNew()`
- `StarDel()`
- `StarDraw()`

- **Pour valider ces fonctions, faire le nécessaire dans le module « app » pour créer, afficher et détruire deux entités « star » sans utiliser à ce stade le module « container » : utiliser deux pointeurs distincts, et des paramètres différents pour chaque entité créée.**

Les emplacements repérés par le type de tag donné ci-dessous sont prévus à ces fins de tests préliminaires en cours de développement de l'application. Les instructions qui y sont placées pour ces tests devront être désactivées par la suite.

```
/** -----FOR TESTING PURPOSE-----
 * @todo Create here two star entities
 */
```


- **Implémenter la fonction suivante du module « star » :**
 - StarMove()
- **Valider la fonction en procédant à l'animation des entités star créées précédemment.**

3.1.2 Décalage en position des entités « star » sur mouvement de la souris : fonctionnalité « d'offseting »

Cette fonctionnalité réalise le décalage en position des entités « star » pour que celles-ci suivent le mouvement de la souris lorsque l'un de ses boutons est maintenu enfoncé. L'effet global obtenu est une impression de déplacement de l'espace dans lequel évoluent les entités.

Principe algorithmique :

- Lors de l'appui sur l'un des boutons de la souris, le flag `ST_APP_MOUSE_BUTTON_DOWN` est positionné à vrai, les coordonnées du pointeur la souris sont mémorisées dans le champ `m_ptStart` de l'application.
 - Sur mouvement de la souris et tant que le bouton est maintenu enfoncé, on calcule le déplacement effectué par la souris depuis le point de départ précédemment mémorisé. Ce déplacement constitue l'offset que l'on applique aux entités « star » présentes dans la simulation.
 - On met à jour le champ `m_ptStart` avec les coordonnées actuelles du pointeur de la souris, pour le prochain déplacement de la souris.
 - Ce scénario est reconduit tant que le bouton reste appuyé.
 - Sur relâchement du bouton, le flag `ST_APP_MOUSE_BUTTON_DOWN` est positionné à faux, le mécanisme d'offset cesse alors.
- **Implémenter la fonction suivante du module « star » :**
 - StarOffset()
 - **Valider cette fonction sur les entités star créées précédemment.**

STS 1 SN IR 2021-2022	PROGRAMMATION STRUCTUREE DS – 4H00	
---------------------------------	--	---

3.2 Module « Container »

A partir de ce stade du développement, les entités « individuelles » « star » créées précédemment à des fins de tests ne devront plus être actives dans l'application : celles-ci seront désactivées en commentant la directive de compilation `#define TEST_ISOLATE_ENTITY_STAR` située en tête du fichier "app.c".

VOUS N'EFFACEREZ PAS LE CODE MIS EN PLACE POUR CES TESTS INDIVIDUELS !

3.2.1 Fonctionnalités de base

➤ **Implémenter les fonctions suivantes du module « container » :**

- NodeNew()
- NodeDelReturnNext()
- ContainerNew()
- ContainerDel()
- ContainerFlush()
- ContainerCard()
- ContainerPushback()
- ContainerParse()

➤ **Valider ces fonctions en y plaçant deux entités « star » et en transposant les appels des fonctions StarDraw(), StarMove() et StarOffset() pour que celles-ci soient traitées à partir du module « container ».**

Le comportement attendu doit être identique à celui obtenu lors des tests individuels menés dans la phase précédente du développement.

Le nombre d'entités « star » présentes dans la simulation sera indiqué dans la barre de titre de la fenêtre de l'application.

La fonctionnalité flush du module « container » pourra être validé sur l'appui de la touche 'F' du clavier.

3.2.2 Fonctionnalité de parsing avec destruction

➤ **Implémenter la fonction suivante du module « container » :**

- ContainerParseDellf()

➤ **Valider cette fonction sur le mouvement des entités « star ». Celles-ci devront être détruites lors du franchissement des limites de la zone représentant l'espace de la simulation.**

La fonction StarMove() devra être adaptée en conséquence si tel n'est pas le cas pour vous à ce stade du développement.

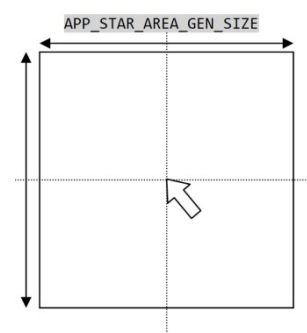
Pour les calculs des débordements de la zone de simulation, on prendra en compte la position du centre de l'entité « star » concernée en y ajoutant ou en y retranchant selon le cas son rayon global.

Pour ce test, la délimitation physique de la zone d'espace devra figurer dans la simulation.

Constater le comportement de l'application pour une valeur négative assez grande du paramètre `APP_SPACE_PADDING` lorsque l'on « déplace » l'espace avec la souris.

3.2.3 Génération de constellations d'étoiles


La génération des étoiles est du ressort de l'application, et cela de manière aléatoire en nombre et en rythme de création. Le paramètre laissé à l'utilisateur, est la région dans laquelle cette génération aura lieu. Cette région est centrée sur le pointeur de la souris et occupe une zone carrée paramétrée par la constante symbolique `APP_STAR_AREA_GEN_SIZE`.



Principe algorithmique :

Dans la fonction d'animation :

- A chaque fois que le champ `m_generateTimer` de l'entité « app » atteint la valeur 0, il est procédé à une nouvelle phase de génération d'étoiles :
 - Le nombre d'étoiles à générer est tiré de manière aléatoire entre `APP_STAR_NB_MIN` et `APP_STAR_NB_MAX` ;
 - Ce nombre d'étoiles est généré avec les paramètres aléatoires suivants, et sont placées dans le conteneur associé :
 - Position aléatoire du centre de l'étoile située dans la zone de génération autour du pointeur de la souris ;
 - Composantes vitesse aléatoires comprises entre `-APP_STAR_SPEED_MAX` et `+APP_STAR_SPEED_MAX`, tout en s'assurant de ne pas avoir ces composantes nulles toutes les deux à la fois ;
 - Nombre aléatoire de branches compris entre `APP_STAR_NB_BRANCHES_MIN` et `APP_STAR_NB_BRANCHES_MAX` ;
 - Rayon aléatoire compris entre `APP_STAR_RADIUS_MIN` et `APP_STAR_RADIUS_MAX` ;
 - Couleur dont les composantes RGB aléatoires sont comprises entre `APP_STAR_RGB_MIN` et `APP_STAR_RGB_MAX` ;
 - La nouvelle valeur du champ `m_generateTimer` est tirée de manière aléatoire entre `APP_STAR_TIMER_GEN_MIN` et `APP_STAR_TIMER_GEN_MAX` ;

STS 1 SN IR 2021-2022	<p>PROGRAMMATION STRUCTUREE</p> <p>DS – 4H00</p>	
---------------------------------	---	---


- Le champ `m_generateTimer`, si sa valeur n'est pas nulle, est décrémenté d'une unité.

➤ **Implémenter la fonctionnalité de génération automatique d'étoiles.**

On pourra matérialiser la zone de création en traçant le carré correspondant à l'écran, et vérifier le déplacement de la zone de génération avec celui de la souris.

Le nombre d'étoiles présentes dans la simulation sera toujours affiché dans la barre de titre de la fenêtre.

On pourra forcer la génération par appui sur la barre ESPACE du clavier.

STS 1 SN IR 2021-2022	PROGRAMMATION STRUCTUREE DS – 4H00	
---------------------------------	--	---

TRAVAIL DEMANDE - RECOMMANDATIONS

UN LECTURE COMPLETE ATTENTIVE DE L'INTEGRALITE DU SUJET EST RECOMMANDEE AVANT D'ENTAMER TOUT DEVELOPPEMENT OU CODAGE.

Une base applicative est fournie sous l'archive « NomPrenom_Base.zip » contenant un projet configuré Eclipse/SDL/GCC.

Après importation sous Eclipse de cette base, renommer aussitôt le projet, comme à l'accoutumée, avec vos <NomPrenom>. Renommer ensuite le fichier principal contenant la fonction main() avec également vos <NomPrenom.c>.

Dans les entêtes des fichiers .h et .c, indiquer vos nom et prénom en lieu et place prévu à cet usage.

Le travail demandé consiste à l'élaboration des codes répondant aux fonctionnalités attendues aux emplacements marqués « ToDo ». **VOUS N'EFFACEREZ PAS CES MARQUES MAIS PLACEREZ LE CODE DEMANDE EN DESSOUS DE CHACUNE D'ELLES.**

Il est également recommandé de procéder par étapes constructives dans l'élaboration de l'application, on la construisant pas à pas, et en testant chaque étape avant d'empiler la suivante.

A REMETTRE - RECOMMANDATIONS

Le projet complété et paramétré avec vos nom et prénom, sera archivé au format .zip par l'utilitaire d'exportation intégré à Eclipse. L'archive devra elle-même être nommée de vos nom et prénom, qui on le rappelle, SANS ACCENTS SANS ESPACES PREMIERE LETTRE DES MOTS EN MAJUSCULE.

Cette archive sera à déposer sous Moodle.