Filière :

Ingénierie logicielle et intégration des systèmes informatiques

RAPPORT PARTIE 1

MODULE : INTÉGRATION DE PROGRAMMATION (PROJET GTK)

ANNÉE UNIVERSITAIRE : 2018 - 2019

RÉALISÉ PAR :
AOUITA SARA
NEKKAS MANAL

ENCADRÉ PAR :
MR. A. BEKKHOUCHA

Table de matière

I-INTRODUCTION :

La bibliothèque GTK+ (The GIMP Toolkit), écrite en C permet de créer une interface graphique pour vos applications.
Vous pouvez créer fenêtres, boutons, menus, barres d'outils, barres de progression, etc... Et il est même possible d'imprimer, d'appliquer un thème, et bien d'autres choses encore !

Quelques bibliothèques de GTK+ :

GLib : Une bibliothèque d'utilité générale, non spécifique aux interfaces utilisateur graphiques. GLib fournit de nombreux types de données utiles, des macros, des conversions de type, des utilitaires de chaîne, des utilitaires de fichiers, une abstraction de boucle principale, etc.

GObject : Une bibliothèque qui fournit un système de types, une collection de types fondamentaux comprenant un type d'objet, un système de signal.

GIO : Une API VFS moderne et facile à utiliser comprenant des abstractions pour les fichiers, les disques, les volumes, les E / S de flux, ainsi que la programmation réseau et la communication DBus.

Cairo : Une bibliothèque graphique 2D prenant en charge plusieurs périphériques de sortie.

Pango : Une bibliothèque pour la gestion de texte internationalisée. Il se centre autour de l'objet PangoLayout, représentant un paragraphe de texte. Pango fournit le moteur pour GtkTextView, GtkLabel, GtkEntry et d'autres widgets qui affichent du texte.

ATK : La boîte à outils d'accessibilité. Il fournit un ensemble d'interfaces génériques permettant aux technologies d'accessibilité d'interagir avec une interface utilisateur graphique. Par exemple, un lecteur d'écran utilise ATK pour découvrir le texte dans

une interface et le lire aux utilisateurs aveugles. Les widgets GTK + ont un support intégré pour l'accessibilité en utilisant le framework ATK.

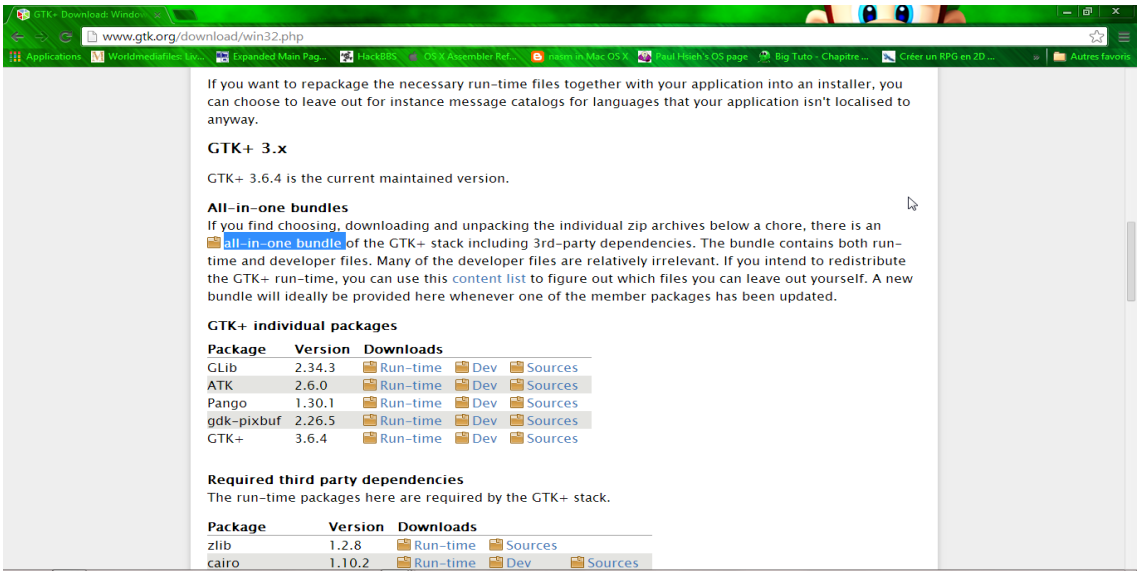
GdkPixbuf : C'est une petite bibliothèque qui vous permet de créer des objets GdkPixbuf ("pixel buffer") à partir de données d'image ou de fichiers image. Utilisez un GdkPixbuf en combinaison avec GtkImage pour afficher des images.

GDK : La couche d'abstraction qui permet à GTK + de prendre en charge plusieurs systèmes de fenêtrage. GDK fournit des fonctionnalités de système de fenêtrage sur X11, Windows et OS X.

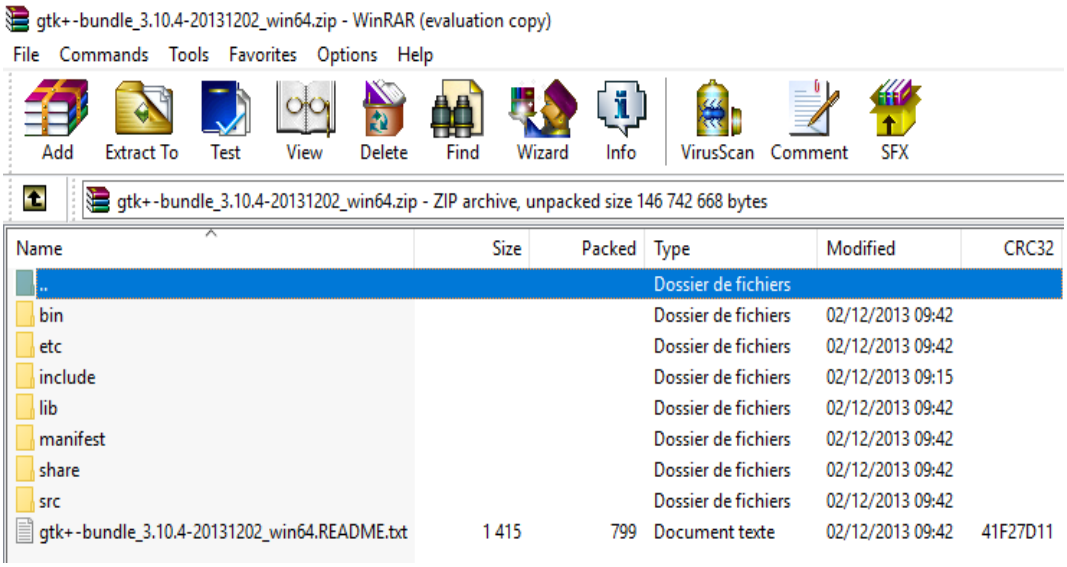
GTK + : La bibliothèque GTK + elle-même contient des widgets, c'est-à-dire des composants graphiques tels que GtkButton ou GtkTextView.

II-INSTALLATION DE GTK+:

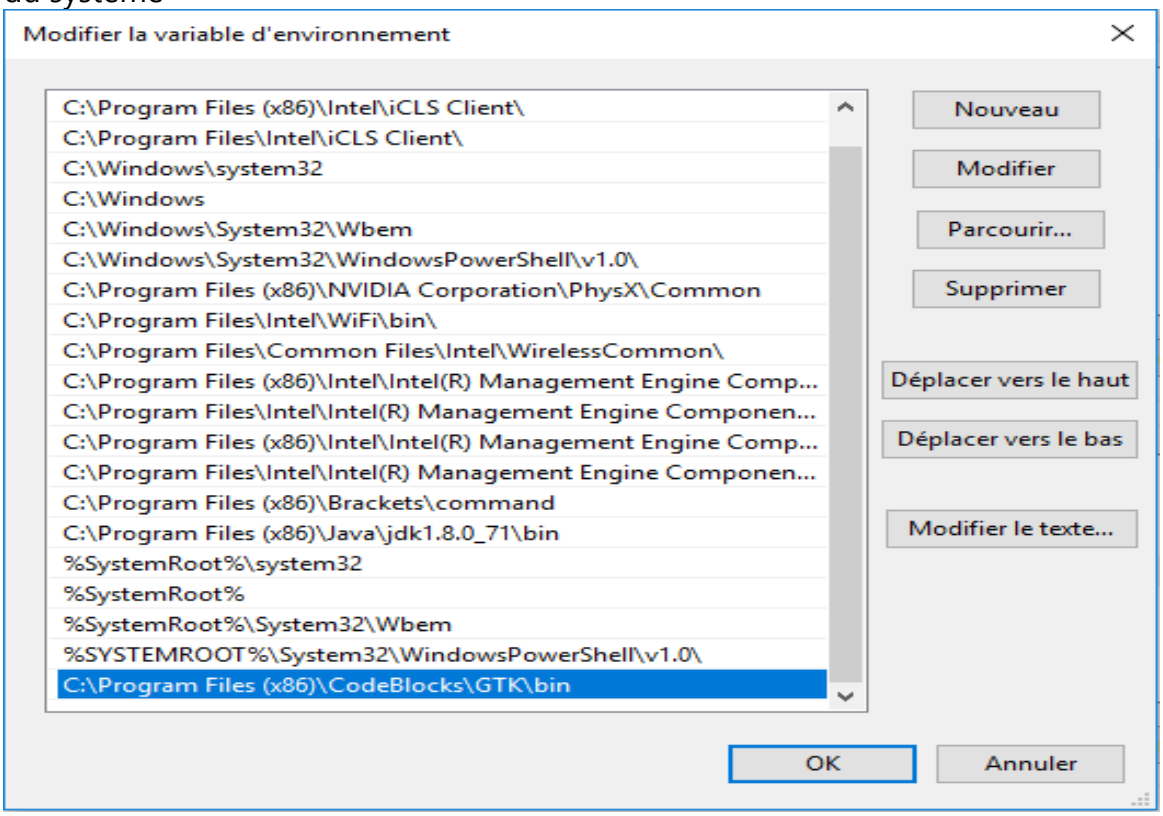
Tout d'abord il faut se rendre sur cette page : http://www.gtk.org/download/
-il faut ensuite suivre le lien suivant son système d'exploitation. Si tu es sous Windows ; alors il faut cliquer sur Windows 32 bits ou 64 bits selon ton système (si tu ignores quel est le type de ton système Windows clique sur l'icône de Windows et clic droit sur "ordinateur" ou "poste de travail" selon ta version et puis choisi "propriété" dans le menu qui s'ouvre. Là tu verras le type de ton système)
-il faut ensuite aller vers le bas juste après gtk+ 3.x et choisir le lien "all-in-one bundle" comme sur la capture suivante.
Voici un lien alternatif pour directement télécharger le "all-in-one-bundle" selon votre système Windows



Après le téléchargement, il faut décompresser le fichier dans un dossier.

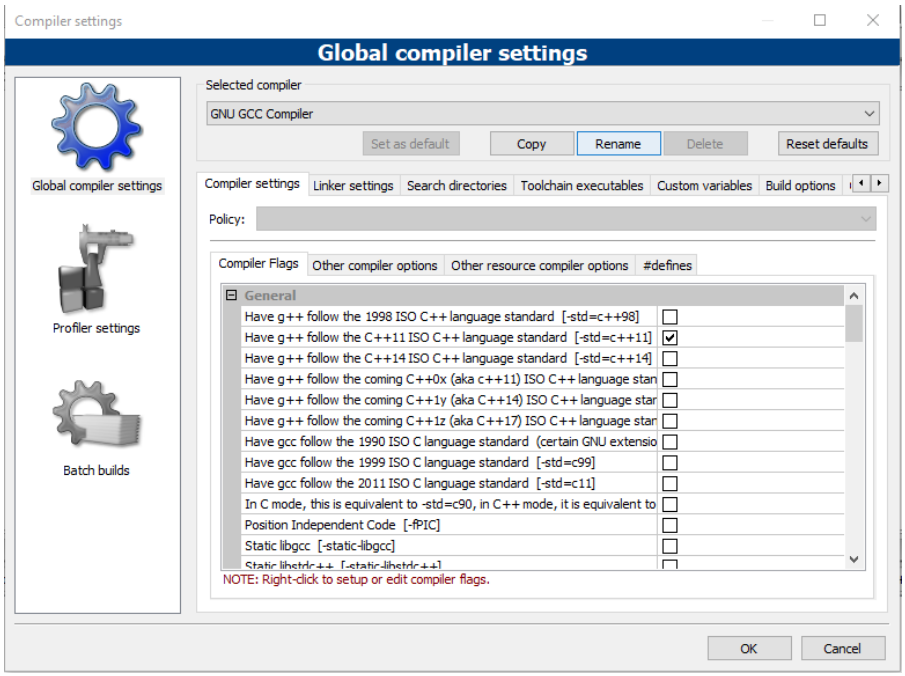


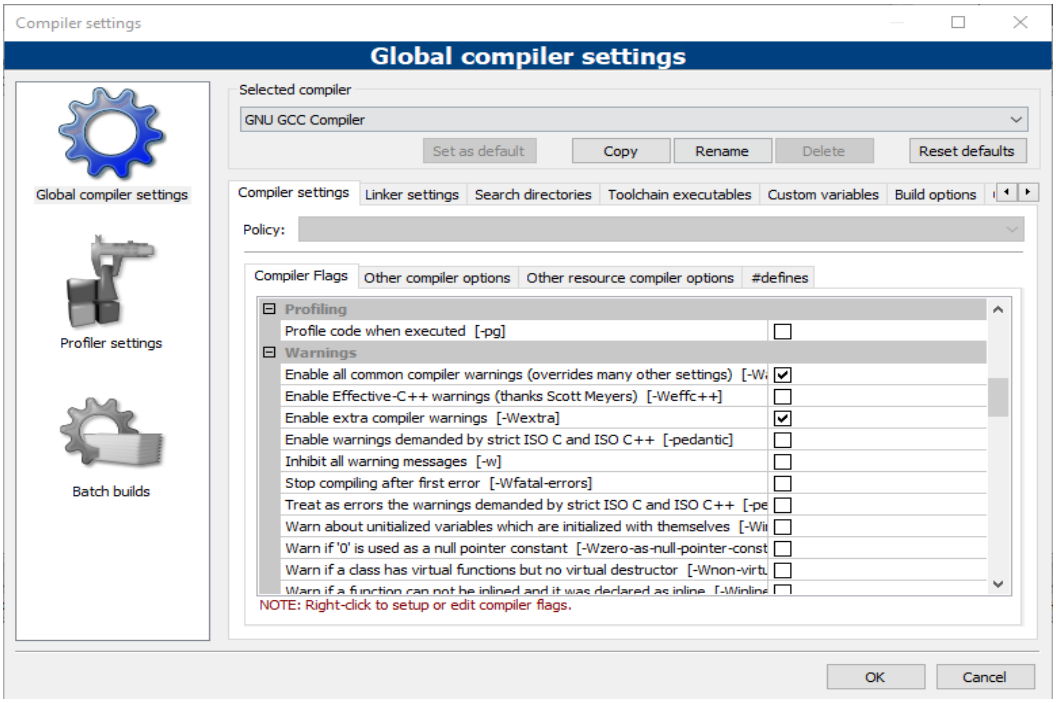
Ensuite, ajouter l'emplacement du fichier bin du répertoire GTK au Path des variables du système



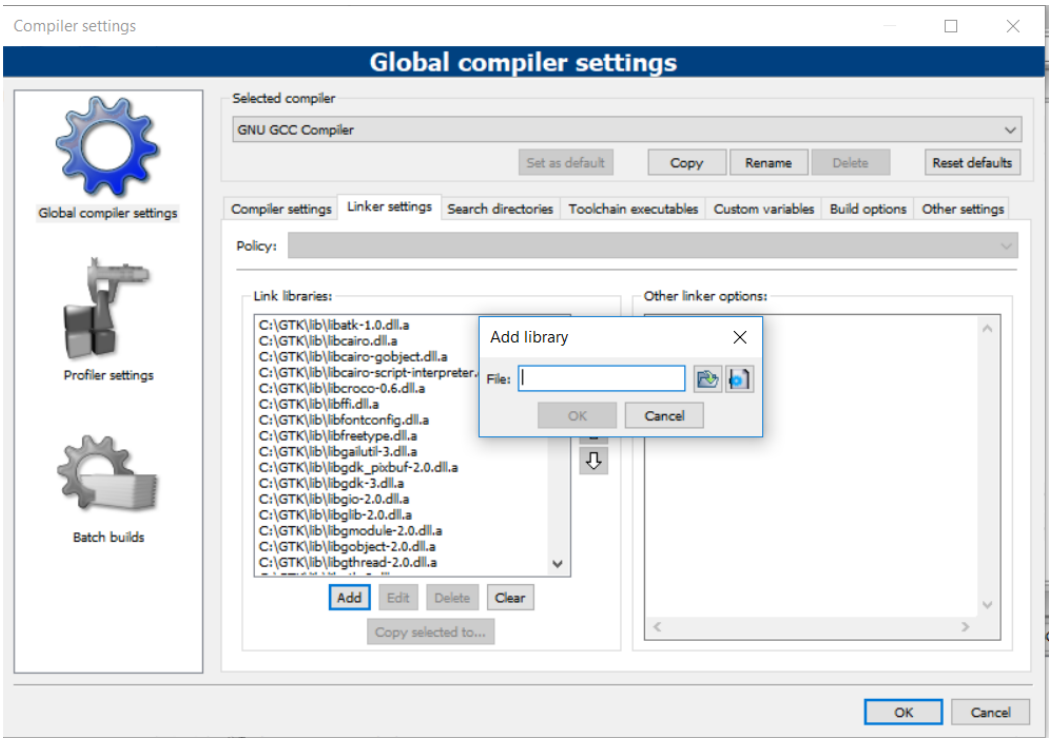
Le point suivant c'est la configuration de Code Blocks.

- D'abord, ouvrir les paramètres du compiler en cliquant sur le bouton « Settings => Compiler »
- Ensuite sur « Compiler settings » exactement au « Compiler Flags », vous cochez ces champs comme l'image vous montre :

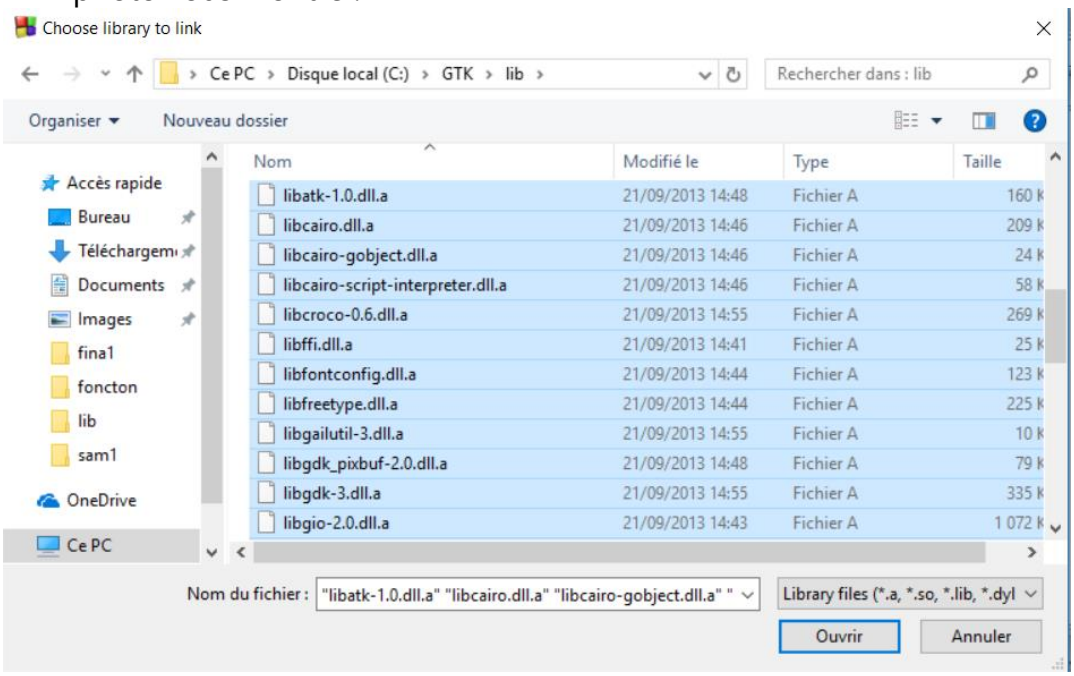




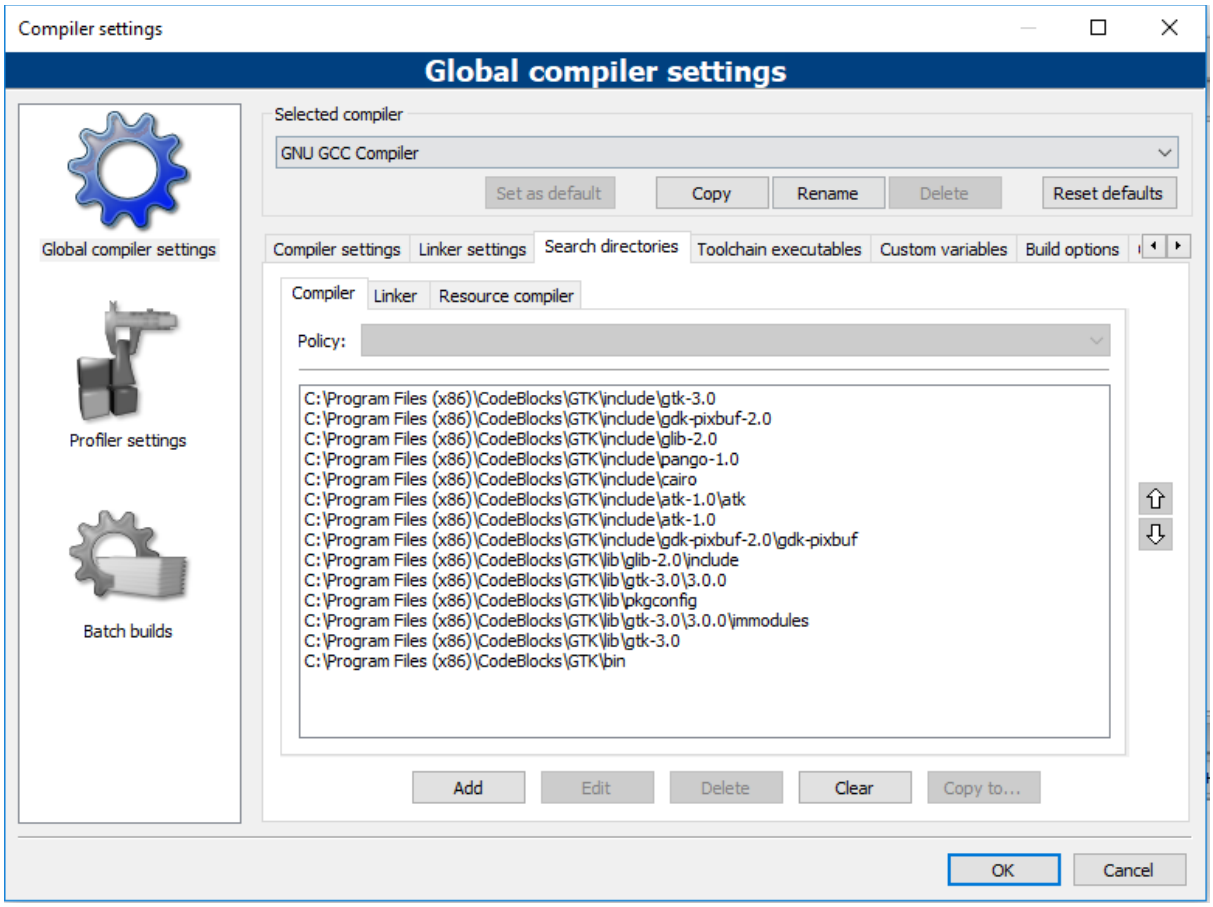
- Passant maintenant au deuxième champ qui représente « Linker settings », vous allez inclure tous les fichiers d'extension '.a' qui sont dans le fichier lib du répertoire GTK, et voilà une démonstration avec des photos que vous devez suivre :
- Il faut cliquer sur « Add »



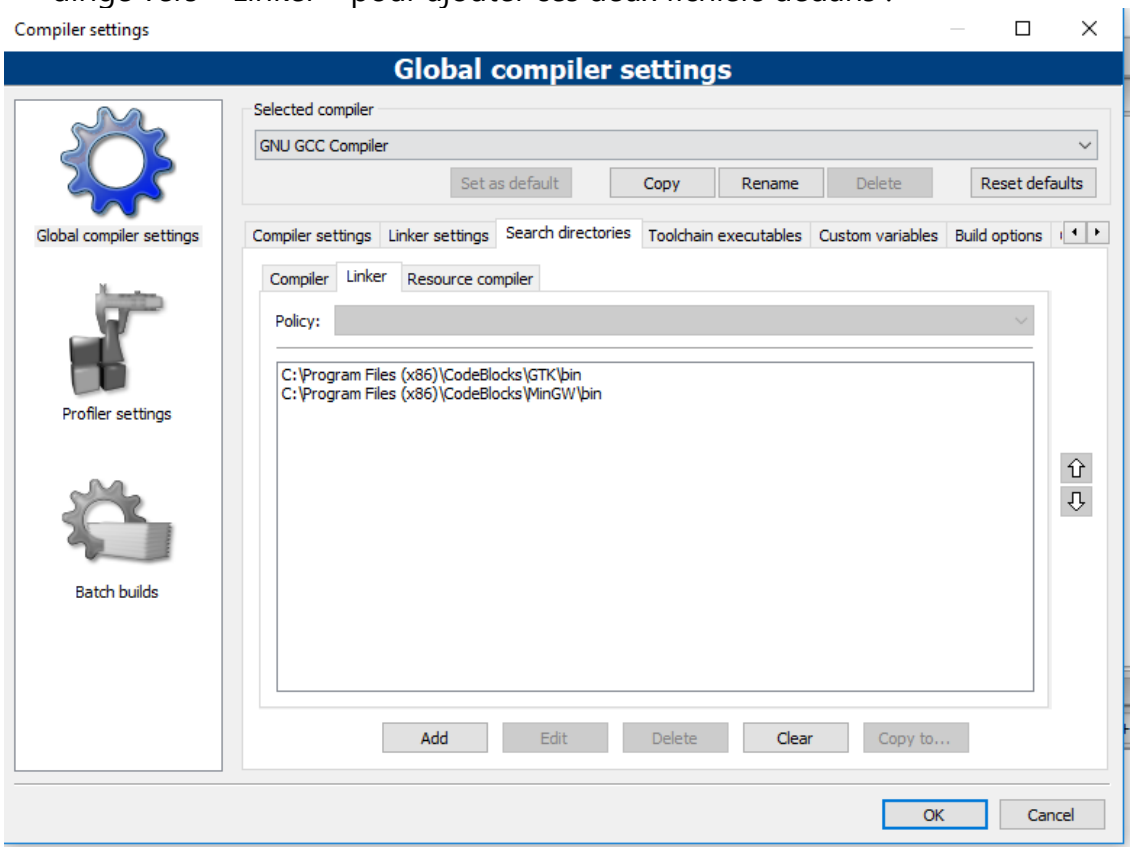
- Cliquer sur le fichier lib pour ajouter les fichiers d'extension '.a' comme la photo vous montre :



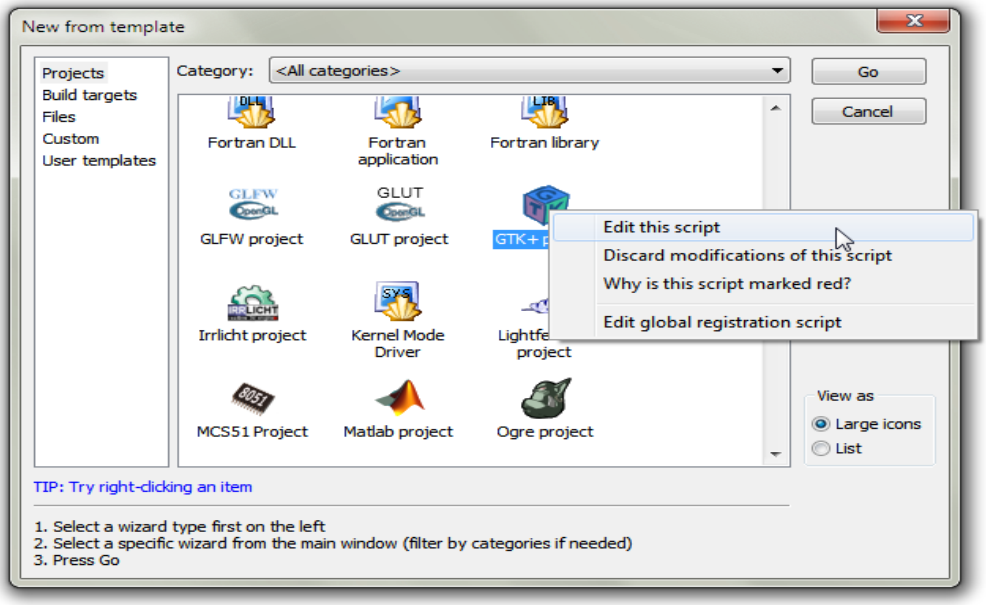
- Le champ suivant c'est « Search directories », on clique sur « Compiler », pour ajouter les dossiers ci-dessous en suivant la même méthode précédente :



❖ Et pour conclure, on reste sur le même champ « Search directories », et on se dirige vers « Linker » pour ajouter ces deux fichiers dedans :



- Après avoir configuré les paramètres de GTK, créer un nouveau projet avec code blocks et dans la liste des projets, venir sur gtk et cliquer droit sur lui et choisir "edit this script" Comme sur la figure suivante :



- Alors une page s'affiche, il faut copier son contenu et le sauvegarder dans un fichier texte (en fait c'est une double sécurité car code blocks prévoit dans son menu, l'option "discard modifications on this script" qui permet de revenir au script d'origine). Il faut ensuite effacer le contenu du script et le remplacer par celui-ci que j'ai créé de toutes pièce car le script de code blocks ne permet pas de gérer la version 3 de gtk+ mais juste la version 2 c'est d'ailleurs pour cela qu'on a fait tout ce détour pour l'installation.

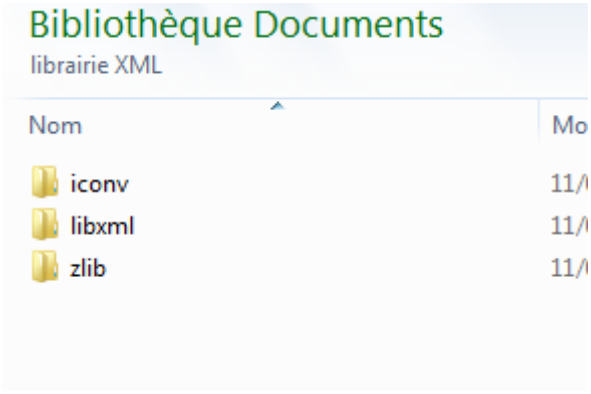
Il faut ensuite l'enregistrer. Il faut alors réessayer de créer un nouveau projet gtk+ mais cette fois une fois que l'assistant de création de projet se présente, il faut juste choisir gtk+ en double-cliquant dessus. Si tout se passe bien l'assistant vous demande le nom de votre projet alors fournissez toutes les infos et enfin cliquez suivant et là une nouvelle boite s'ouvre cliquez encore suivant et une autre boite s'ouvre vous indiquant qu'il ne sait pas où se trouve gtk sur votre ordinateur.

Alors vous allez dans le champ base et renseignez le dossier où vous aviez décompressé le contenu de votre zip "all-in-one-bundle" précédemment téléchargé.
Ensuite vous faites la même chose pour les champs en indiquant le dossier include qui se trouve dans votre dossier décompressé pour le champ include
Et vous faites la même chose pour le champ lib en indiquant le chemin qui mène également à votre dossier lib lui aussi situé dans votre dossier décompressé. Et voilà un projet s'ouvre avec un fichier main.c qui contient du code compilez le et voyez le résultat.

III-Installation de XML :

XML est un métalangage qui vous permet de concevoir votre propre langage à balises. Un langage à balises classique définit une manière de décrire des informations dans une certaine classe de documents (ex : HTML). XML vous laisse définir votre propre langage à balises pour plusieurs classes de documents. Il vous permet cela car il est écrit en SGML, le métalangage standard international pour les langages à balises.
Pour l'installation de la bibliothèque XML, il suffit de suivre les étapes suivantes :

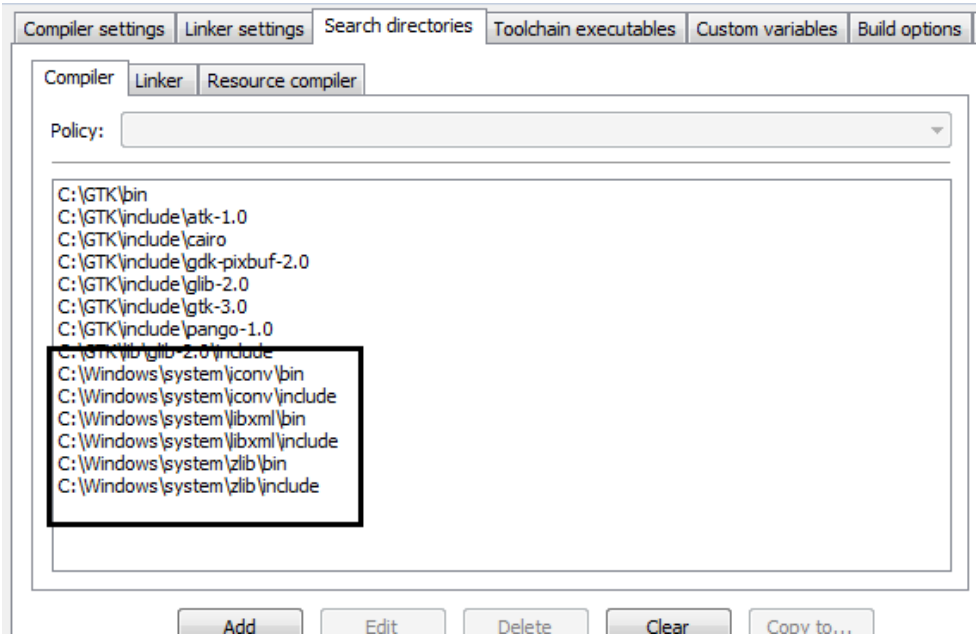
a-Télécharger le Paquet des librairies XML:



Le Paquet XML.

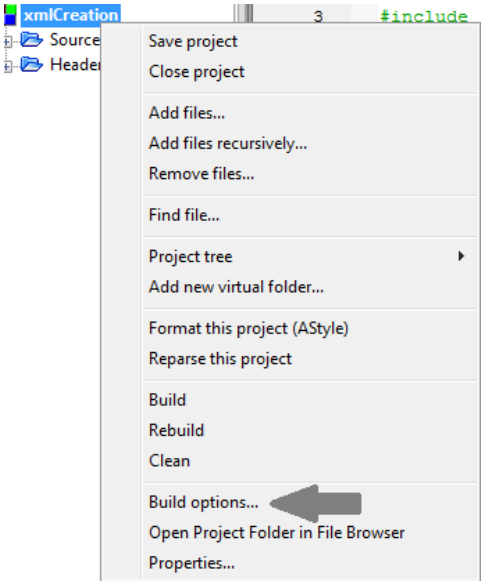
b-Configuration de Code: Blocks IDE:

Après avoir lancé votre IDE, cliquez sur Setting dans la barre de menus après sur Compiler après sur Search Directories et enfin sur Compiler, et inclure les fichiers suivant un par un.

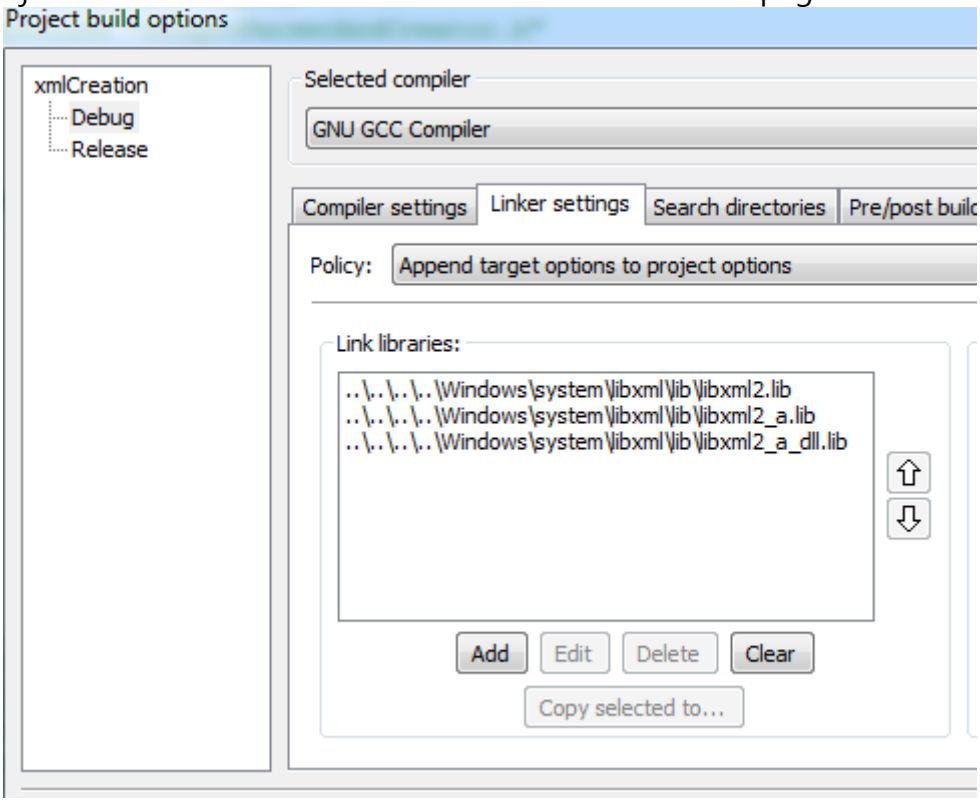


c-Configuration du projet:

-Menu contextuel :



Et enfin, ajouter le lien du dossier lib du dossier libxml dans la page Linker Setting.



Linker Setting.

2- ***Bibliothèque LibXML2:***

La bibliothèque libxml est très riche en fonctionnalités, elle est un analyseur et une boîte à outils XML C développés pour le projet Gnome (mais utilisables en dehors de la plate-forme Gnome). Il s'agit d'un logiciel libre disponible sous la licence MIT . XML lui-même est un métalangage pour la conception de langages de balisage, c'est-à-dire un texte où la sémantique et la structure sont ajoutées au contenu en utilisant des informations de "balisage" supplémentaires placées entre crochets. HTML est le langage de balisage le plus connu. Bien que la bibliothèque soit écrite en C, diverses liaisons de langue la rendent disponible dans d'autres environnements.

Libxml2 est connu pour être très portable, la bibliothèque doit être construite et fonctionner sans problèmes sérieux sur une variété de systèmes (Linux, Unix, Windows, CygWin, MacOS, Mac OS X, OS RISC, OS / 2, VMS, QNX, MVS, VxWorks , ...)

3-Noeud et traitement :

En fait, il existe deux méthodes pour parser un document XML :

- Document Object Model : la méthode DOM consiste à transformer un document XML sous forme d'un arbre en mémoire. Cette méthode convient donc parfaitement aux fichiers de petite taille : la modification du document XML est beaucoup plus facile et rapide mais est coûteux en termes de temps, performance et utilisation de la mémoire. Elle permet également l'utilisation d'expressions XPath, pour chercher un ou des nœuds, par exemple.
- Simple Api for Xml : La méthode SAX consiste à déclencher un événement pour les langages orientés objets ou en l'appel de fonctions de rappel lorsque le parser rencontre un élément XML spécifique comme un début de balise, un commentaire, une fin de balise, ... Contrairement à DOM, SAX permet de parcourir un document XML sans le conserver en mémoire. Elle est donc particulièrement adaptée aux documents XML très volumineux ou pour procéder à des traitements simples.

4- fonctionnalité de XML :

La bibliothèque libxml2 agit sur un type de pointeur précis, xmlNodePtr, pointeur sur le nœud de l'arbre DOM, et le pointeur sur le document XML, xmlDocPtr.

On citera les quelques fonctions qui nous ont servies, dans les traitements, sauvegarde et chargement du fichier XML :

On déclare le nœud racine, et un pointeur de type document, puis on les crée avec les fonctions suivantes :

On associe la racine au document grâce à la fonction :

```
Void xmlDocSetRootElement (xmlDocPtr, xmlNodePtr);
```

Pour le récupérer, on appelle la fonction :

```
xmlNodePtr* xmlDocGetRootElement(xmlDocPtr);
```

Pour ajouter une propriété à une balise on appelle la fonction :

```
xmlSetProp (xmlNodePtr node, const xmlChar * name,const xmlChar * value)
```

Pour la récupérer :

```
xmlChar * xmlGetProp (const xmlNode * node,const xmlChar * name)
```

5- XML parser :

Un analyseur syntaxique XML (ou parseur), permet de récupérer dans une structure XML, des balises, leur contenus, leurs attributs et de les rendre accessibles.
Il existe deux types de parseurs :
les analyseurs non-validant, qui contrôlent simplement que le document XML est bien formé.
les analyseurs validant, qui vérifient que le document XML est valide.

6-les Fonction utilisé dans notre programme :

```
#include "xmlParser.h"

xmlDoc* chargerFichierXML(char cheminFichier[200])
{
    //  Charge le contenu du fichier XML dans la memoire
    xmlDoc *doc = xmlReadFile(cheminFichier,NULL,0);

    //  Verifier si le fichier a bien été chargé
    if(doc == NULL)
    {
        printf("Le fichier contenant les propriétés n'a pu être chargé");
        exit(1);
    }

    return doc;
}
```

```
xmlDoc* chargerFichierXML(char cheminFichier[200])
{
   //     Charge le contenu du fichier XML dans la memoire
          xmlDoc *doc = xmlReadFile(cheminFichier,NULL,0);

          //          Verifier si le fichier a bien été chargé
          if(doc == NULL)
          {
                    printf("Le fichier contenant les propriétés n'a pu être chargé");
                    exit(1);
          }

          return doc;
}
```

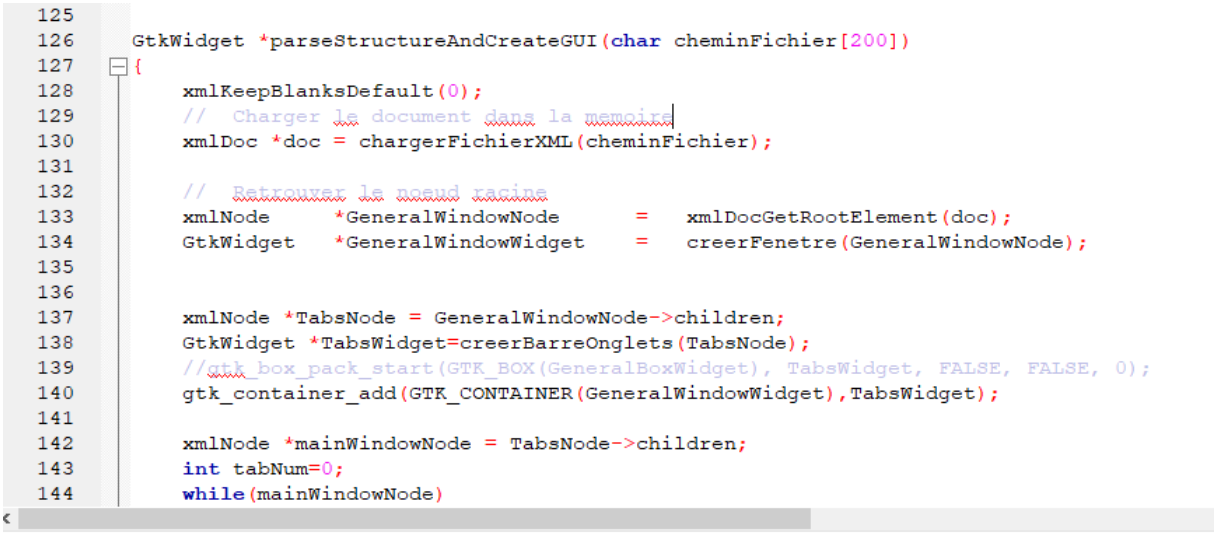
```c
#include "widgetParserAndCreator.h"


GtkWidget *createWidget(GtkWidget* parentWidget,xmlNode *node,GtkWidget *GeneralWindowWidget)
{
    GtkWidget *widget = NULL;

    if(xmlStrcmp(node->name, "window") == 0)
    {
        widget = creerFenetre(node);
    }
    else if(xmlStrcmp(node->name, "box") == 0)
    {
        widget = creerBox(node);
    }
    else if(xmlStrcmp(node->name, "menubar") == 0)
    {
        widget = creerBarreMenu(node);
    }
    else if(xmlStrcmp(node->name, "menu") == 0)
```

```
GtkWidget *createWidget (GtkWidget* parentWidget,xmlNode *node,GtkWidget
*GeneralWindowWidget)
{
GtkWidget *widget = NULL;

if(xmlStrcmp(node->name, "window") == 0)
{
widget = creerFenetre(node);
}
//  Une petite verification avant de retourner le widget
if(widget)
return widget;
else
{
printf("widget %s n'a pas pu etre cree, le programme s'arretera",node->name);
exit(0);
}
```



```
125
126  GtkWidget *parseStructureAndCreateGUI(char cheminFichier[200])
127  {
128      xmlKeepBlanksDefault(0);
129      //  Charger le document dans la memoire
130      xmlDoc *doc = chargerFichierXML(cheminFichier);
131
132      //  Retrouver le noeud racine
133      xmlNode     *GeneralWindowNode      =   xmlDocGetRootElement(doc);
134      GtkWidget   *GeneralWindowWidget    =   creerFenetre(GeneralWindowNode);
135
136
137      xmlNode *TabsNode = GeneralWindowNode->children;
138      GtkWidget *TabsWidget=creerBarreOnglets(TabsNode);
139      //gtk_box_pack_start(GTK_BOX(GeneralBoxWidget), TabsWidget, FALSE, FALSE, 0);
140      gtk_container_add(GTK_CONTAINER(GeneralWindowWidget),TabsWidget);
141
142      xmlNode *mainWindowNode = TabsNode->children;
143      int tabNum=0;
144      while(mainWindowNode)
```

```
GtkWidget *parseStructureAndCreateGUI(char cheminFichier[200])
{
   xmlKeepBlanksDefault(0);
   //  Charger le document dans la memoire
   xmlDoc *doc = chargerFichierXML(cheminFichier);

   //  Retrouver le noeud racine
   xmlNode     *GeneralWindowNode      =   xmlDocGetRootElement(doc);
   GtkWidget   *GeneralWindowWidget    =   creerFenetre(GeneralWindowNode);


   xmlNode *TabsNode = GeneralWindowNode->children;
   GtkWidget *TabsWidget=creerBarreOnglets(TabsNode);
   //gtk_box_pack_start(GTK_BOX(GeneralBoxWidget), TabsWidget, FALSE,
FALSE, 0);
   gtk_container_add(GTK_CONTAINER(GeneralWindowWidget),TabsWidget);

   xmlNode *mainWindowNode = TabsNode->children;
   int tabNum=0;
   while(mainWindowNode)
   {
      //  Verifier si le noeud racine est une fenetre
      if(xmlStrcmp(mainWindowNode->name,"tab") == 0)
      {
         //  Creer le widget fenetre
         GtkWidget* mainWindow = creerBox(mainWindowNode);

         GtkWidget *scrollbar = gtk_scrolled_window_new(NULL, NULL);
         gtk_scrolled_window_set_policy(scrollbar,1,1);


gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(scrollbar),
mainWindow);
```

```
            creerOnglet(TabsWidget,scrollbar,mainWindowNode);
            //  Trouver le noeud fils de la fenetre, le creer et le lier à son parent
           // g_signal_connect(G_OBJECT(mainWindow), "button-press-event",
         //   G_CALLBACK(show_popup), mainWindowNode->name);

            printf("Parent: %s Child: %s\n", mainWindowNode->name,
mainWindowNode->children->name);
            if(xmlStrcmp(mainWindowNode->children->name,"text"))
               createAndLinkChild(mainWindow,1,mainWindowNode->children,
"box",GeneralWindowWidget);
         }
         mainWindowNode=mainWindowNode->next;
      }

      return  GeneralWindowWidget;
}

//  Child node is the current node
void createAndLinkChild(GtkWidget *parentWidget, int parentNature, xmlNode
*childNode, char *parentName,GtkWidget *GeneralWindowWidget)
{
      xmlKeepBlanksDefault(0);
      printf("-------------------------------------------------\n\n");
      printf("Entrain de creer: %s\n", childNode->name);
      //  Verifier si on est arrivé à un noeud feuille (pas de noeud(s) enfant(s))
      if(childNode && (xmlStrcmp(childNode->name,"text"))!=0)
      {
         int top, left, height, width;

         //  Creer le widget et le retourner
         GtkWidget *childWidget =
createWidget(parentWidget,childNode,GeneralWindowWidget);

         //  Le lier à son widget pere
         switch(parentNature)
         {
            case 1:
               gtk_container_add(GTK_CONTAINER(parentWidget),childWidget);
               printf("Linked %s to %s\n", childNode->name, parentName);
               break;
            case 2:
               gtk_menu_shell_append(GTK_MENU_SHELL(parentWidget),
childWidget);
               printf("Linked %s to %s\n", childNode->name, parentName);
               break;
            case 3:

gtk_toolbar_insert(GTK_TOOLBAR(parentWidget),GTK_TOOL_ITEM(childWidget
),0);
```

```
            printf("Linked %s to %s\n", childNode->name, parentName);
            break;
        case 4:
            top = atoi(xmlGetProp(childNode,"startCol"));      //  La colonne d'où la
cellule va débuter
            left = atoi(xmlGetProp(childNode,"startRow"));     //  La ligne d'où la cellule
va débuter
            height = atoi(xmlGetProp(childNode,"finishRow"));  //  La ligne où la
cellule va se terminer
            width = atoi(xmlGetProp(childNode,"finishCol"));   //  La colonne où la
cellule va se terminer

gtk_grid_attach(GTK_GRID(parentWidget),childWidget,left,top,width,height);
            printf("Linked %s to %s\n", childNode->name, parentName);
            break;
        case 5:
            gtk_box_pack_start(GTK_BOX(parentWidget), childWidget, FALSE,
FALSE, 0);
            printf("Linked %s to %s\n", childNode->name, parentName);
            break;
        case 6:

gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(parentWid
get),childWidget);
            break;
        case 7:
            top = atoi(xmlGetProp(childNode,"startCol"));      //  La colonne d'où la
cellule va débuter
            left = atoi(xmlGetProp(childNode,"startRow"));     //  La ligne d'où la cellule
va débuter
            height = atoi(xmlGetProp(childNode,"finishRow"));  //  La ligne où la
cellule va se terminer
            width = atoi(xmlGetProp(childNode,"finishCol"));   //  La colonne où la
cellule va se terminer

gtk_table_attach_defaults(GTK_TABLE(parentWidget),childWidget,top,width,left,hei
ght);
            printf("Linked %s to %s\n", childNode->name, parentName);
            break;
    }

    printf("J'ai fini de creer: %s\n", childNode->name);
    printf("-------------------------------------------------\n\n");
    //  Si le widget parent a plusieurs widgets fils, il faudra les creer aussi
    if((childNode->next != NULL) && (xmlStrcmp(childNode->next->name,"text")
!= 0))
        createAndLinkChild(parentWidget,parentNature,childNode->next,
parentName,GeneralWindowWidget);

    //  Si le widget courant a un fils, on le cree
```

```
        if((childNode->children != NULL) && (xmlStrcmp(childNode->children-
>name,"text")!=0))

createAndLinkChild(childWidget,atoi(xmlGetProp(childNode,"nature")),childNode-
>children,(char *)childNode->name,GeneralWindowWidget);
    }
}
```

IV-manipulation du widget

Un objet dans GTK est dit widget, la déclaration se fait par : **GtkWidget * objet.**

1-L'INITIALISATION :

L'initialisation de GTK+ se fait par la fonction suivante :
void gtk_init(int* argc, char* argv);**
Les paramètres à passer à cette fonction correspondent aux paramètres reçus à la fonction **main().**

2-L'AFFICHAGE :

Pour tous les widgets utilisés une fonction d'affichage est commune :
Void gtk_widget_show (GtkWidget *widget);

3-LA BOUCLE EVENEMENT

Void gtk_main(void);
Cette fonction crée la boucle évènementielle infinie.
Ces fonctions sont le nécessaire pour la création des applications GTK.

4-LES SIGNAUX

Elle ne servira à rien une interface ornée par toute sorte d'objets, quelques objets peuvent émettre des signaux tel les boutons, tout type de boutons et les événements box qui, en fait mène à ce que la majorité des widgets deviennent dynamique.

5-GÉRER LES ACTIONS

g_signal_connect(gpointer *object, const gchar *name, GCallback func,gpointer func_data);
- *Premier paramètre :* doit être le Widget dont on veut gérer un événement,
- *Deuxième paramètre* : le nom du signal que l'on souhaite intercepter dans une chaine de caractères. Il existe un nom pour chaque signal
- *Troisième paramètre* : nom d'une fonction CALLBACK qui sera appelée lors de l'événement,
- *Dernier paramètre* : on donne un paramètre passé automatiquement à la fonction donnée juste avant.

Quand un objet émet un signal, GTK+ appelle la fonction func avec les paramètres object et func_data. Le prototype de la fonction doit alors être toujours :
void *ma_fonction*(GtkWidget *widget, gpointer data);

6-LES ÉVÉNEMENTS :

Il existe sous GTK une multitude d'évenements qu'on peut connecter à des Widgets, qui permettent d'enclencher : une fonction en réaction à tel ou tel événement produit.
Voici une liste de ceux les plus courants :
- Quand on appuie sur le bouton de la souris sur le Widget :
button_press_event
- Quand on relâche le bouton de la souris :
button_release_event
- Quand on effectue un scroll sur le Widget :
scroll_event
- Quand on déplace la souris sur le Widget :
motion_notify_event
- Quand on cherche à détruire le Widget (par exemple via la croix en haut à droite, ou bien gtk_widget_destroy) :
destroy_event
- Quand on appuie sur une touche du clavier :
key_press_event
- Quand on entre avec la souris dans la Widget :
enter_notify_event
- Quand on sort la souris du Widget :
leave_notify_event
Note importante concernant l'event destroy :
g_signal_connect (window, « destroy_event", gtk_main_quit, NULL);
Ainsi, en appelant la fonction **gtk_main_quit**, on quitte la boucle **gtk_main,** et le programme se termine.

7-fenêtre

Une fenêtre comme n'importe quel autre objet GTK se crée par appel d'une fonction de la bibliothèque. Dite GtkWindow c'est l'élément de base, le support qui contiendra les autres objets, grâce à un objet dit fixed qu'on découvrira après.
Une interface GTK (fenêtre) peut être caractérisée par : son titre, taille, position, position de départ prédéfinie, le type, l'icône, une image de fond, une couleur de fond et une possibilité de redimensionnement.
La fonction de création retourne GtkWidget et reçoit le type de la fenêtre:
GtkWidget* gtk_window_new (GtkWindowType type);
Deux valeurs possibles pour le type :

- GTK_WINDOW_TOPLEVEL : fenêtre classique avec toutes ses décorations
- GTK_WINDOW_POPUP : fenêtre « Pop Up ».

Titre de la fenêtre :

Void gtk_window_set_title (GtkWindow* pFenetre, const gchar* strTitle);

Le gchar n'est autre que le char dans la bibliothèque GTK.

Taille de la fenêtre :

void gtk_window_set_default_size(GtkWindow* pFenetre, gint width, gint height);

Pareil pour le gint, il désigne le type int dans la bibliothèque.

Position de départ par défaut :

On énumère les positions prédéfinies :
- GTK_WIN_POS_NONE : la fenêtre est placée n'importe où.
- GTK_WIN_POS_CENTER : la fenêtre est centrée sur l'écran.
- GTK_WIN_POS_MOUSE : la fenêtre est placée là où se trouve la souris à la création de la fenêtre.
- GTK_WIN_POS_CENTER_ALWAYS : la fenêtre est toujours au centre de l'écran.
- GTK_WIN_POS_CENTER_ON_PARENT : la fenêtre est centrée sur sa fenêtre parente.

Raisonnement :

Void gtk_window_set_resizable (GtkWindow *window, gboolean resizable);

Quand la valeur de la variable resizable est à faux, c'est que l'utilisateur ne pourra pas redimensionner la fenêtre.

Void gtk_container_add (GtkContainer* fenetre, GtkWidget* objet);

La fenêtre est le conteneur, on peut lui coller n'importe quel autre objet de type pointeur GtkWidget.

8-Bouton

Les boutons normaux, eux aussi sont des types, pour un bouton simple vide, la fonction de création a pour prototype :

GtkWidget* gtk_button_new ();

Le constructeur avec libellé, qui crée un bouton avec libellé :

GtkWidget* gtk_button_new_with_label(const gchar* label);

Celui avec une image ou dite icône du stock :

GtkWidget* gtk_button_new_from_stock(const gchar* stock_id);

L'argument stock_id peut prendre comme valeur toute constante faisant référence à une image existante, Il y a une liste importante des images dans le Stock, quelques exemples :
- GTK_STOCK_DIALOG_WARNING : Triangle avec point d'exclamation suivi du texte « Warning ».
- GTK_STOCK_YES : Disque suivi du texte « Yes » avec un raccourci clavier ALT+Y.

- GTK_STOCK_QUIT : Icône et texte « Quitter » avec raccourci clavier ALT+Q.

Le constructeur avec raccourci du clavier :

GtkWidget* gtk_button_new_with_mnemonic(const gchar* label);

Le bouton porte l'étiquette donnée en argument. Elle doit contenir le caractère Souligné ('_'). Le caractère qui suit le souligné permet d'activer le bouton par raccourci clavier.

Void gtk_button_set_relief (GtkButton *button, GtkReliefStyle newstyle);

Définir le style de relief du bouton, qui peut prendre trois valeurs :

- GTK_RELIEF_NORMAL
- GTK_RELIEF_HALF
- GTK_RELIEF_NONE

9-Bouton radio

Les boutons radio sont toujours utilisés en groupe (donc deux boutons radio au minimum) puisque leur objectif est de permettre à l'utilisateur de choisir une, et une seule, option parmi plusieurs possibles.

GtkWidget* gtk_radio_button_new_from_widget(GtkRadioButton *group);

GtkWidget* gtk_radio_button_new_with_label_from_widget(GtkRadioButton *group, const gchar*label);

GtkWidget*gtk_radio_button_new_with_mnemonic_from_widget(GtkRadioButton *group, constgchar *label);

Ce qui caractérise ce type des boutons des autres c'est qu'il agit en groupe,dès la création, le bouton radio est automatiquement ajouté au groupe.

Pour récupérer le label du bouton qui est actif, il faut utiliser les fonctions :

GSList* gtk_radio_button_get_group(GtkRadioButton *radio_button);

Cette fonction permet de récupérer le groupe auquel appartient un bouton radio.

10-Menu

Nous allons cette fois voir quelles sont les solutions que GTK+ met à notre disposition pour ajouter un menu à une fenêtre.

Création de l'élément GtkMenuBar :

GtkWidget* gtk_menu_bar_new(void);

Création de l'élément GtkMenu :

GtkWidget* gtk_menu_new(void);

Création de l'élément GtkMenuItem :

GtkWidget* gtk_menu_item_new_with_label(const gchar* label);

Maintenant que l'élément est créé, il faut l'insérer dans le menu.

void gtk_menu_shell_append(GtkMenuShell *menu_shell, GtkWidget *child);

void gtk_menu_shell_prepend(GtkMenuShell *menu_shell, GtkWidget *child);
Il suffit d'utiliser une des fonctions de création du widget GtkMenuItem .
La première fonction ajoute les éléments de haut en bas alors que la seconde les ajoute de bas en haut.
Il faut maintenant dire que lorsque l'utilisateur cliquera sur l'élément crée, il faudra ouvrir le menu qui a été créé précédemment.
La fonction à utiliser est la suivante :
void gtk_menu_item_set_submenu(GtkMenuItem *menu_item, GtkWidget*submenu);
Dernière étape, on ajoute le sous-menu au menu principal. Pour ce faire, nous utilisons la fonction gtk_menu_shell_append .

11-Les onglets

L'objet en question est : GtkNotebook
Création du widget :
GtkWidget* gtk_notebook_new(void);
Insertion des pages :
Void gtk_notebook_append_page(GtkNotebook*notebook,GtkWidget*Child, GtkWidget*tab_label);
La fonction reçoit l'objet notebook dans lequel, la page sera insérée ainsi que l'objet child : l'objet inséré dans la page, et le widget qui sera affiché dans l'onglet.
Gestion des pages :
- *Navigation :*
Void gtk_notebook_next_page(GtkNotebook*notebook);
Void gtk_notebook_prev_page(GtkNotebook*notebook);
Void gtk_notebook_set_current_page(GtkNotebook*notebook,gint page_num);
- *Gestion des labels:*
GtkWidget* gtk_notebook_get_tab_label(GtkNotebook*notebook,GtkWidget* child);
- *Propriétés des onglets :*
Void gtk_notebook_pset_tab_pos(GtkNotebook*notebook,GtkPositionType pos);

12-Boite de dialogue

Une boite de dialogue se compose généralement de trois objets :
- *Boite de saisie :* GtkWidget* entry
- *Séparateur* : GtkHSeparator
- *Un container* :GtkHBox ou si on travaille avec les positions en coordonnée GtkWidget* Fixed

Affichage :
Pour l'affichage :
Gint gtk_dialog_run(GtkDialog*dialog)

13- Les labels

Un libellé peut être standard, comme il peut être sous un format défini, pour cela il a fallu séparer la structure qui permet d'associer le style du label.

```
typedef struct style
{
char tPolice[40]; //pour definir la police utilsee
char tSize[4];//pour definir la taille du text
char tStyle[20];//pour definir le style du text
char tWeight[20];//pour definir l'epaisseur du text
char tColor[10];//pour definir la couleur du text
char fColor[10];//pour definir la couleur du fond
char tUnderline[10];//pour le soulignement du text
char tBarer[10];//pour barrer le text
}Style;
```

Le choix du type chaine de caractère, facilite en quelque sorte l'insertion de la donnée dans la balise, vu que ce formatage se fait par balise.
Venant au label, il s'agit d'un objet de type GtkWidget aussi, un texte et une position.

```
typedef struct label
{
GtkWidget* idLabel;
char * lTexte; //text du label
int lJustify;// LEFT, RIGHT, CENTER
Position lPos; //position du label
Style lFormat; //style du label
int lStyle;// utiliser un format ou pas
}Label;
```

14-Bouton spin

Un bouton spin est un sélecteur numérique, permet à l'utilisateur de cliquer sur l'une des deux flèches pour incrémenter ou décrémenter la valeur affichée.

```
typedef struct spin
{
GtkWidget *idSpin; //identifiant du bouton spin
gdouble sMin; // valeur maximal pouvant être sélecionnée
gdouble sMax; //valeur miniamel pouvant être sélectionnée
gdouble sPas; // le pas
Position sPos; //la position
Taille sTaille; //la taille
}BoutonSpin;
```

15-Le sélecteur de couleur

Le sélecteur de couleur est un widget composite permet à l'utilisateur de choisir une couleur en manipulant des triplets RVB (Rouge, Vert, Bleu) ou TSV (Teinte, Saturation, Valeur). Chacune des valeurs peut être ajustée à l'aide de gradateurs ou de champs de saisie, ou bien en désignant directement la couleur désirée dans une roue de

teinte-saturation/barre de valeur. Il est également possible de définir l'opacité de la couleur.
"color_changed" est émis à chaque modification de la couleur courante dans le widget, qu'elle vienne de l'utilisateur ou bien d'une définition explicite par la méthode **set_color().**

selectcouleur = gtk.ColorSelection()

Le sélecteur de couleur permet de régler l'opacité d'une couleur (aussi appelée canal alpha). Cette option est désactivée par défaut. Pour autoriser les réglages d'opacité, il faut appeler la méthode ci-dessus en donnant la valeur TRUE à son argument has_opacity. Inversement, l'appeler avec FALSE interdira cette possibilité.

selectcouleur.set_current_color(color)

selectcouleur.set_current_alpha(alpha)

On peut définir explicitement la couleur courante en appelant la méthode **set_current_color()** avec une **GdkColor**.

16-Box

Les box permettent une meilleure organisation des widgets, au détriment d'une limitation en ce qui concerne le positionnement de ces derniers. Il existe deux types de boxs, vertical ou bien horizontal.

Pour créer un box horizontal :

GtkWidget* gtk_hbox_new(gboolean homogeneous, gint spacing);

Pour créer un box vertical :

GtkWidget* gtk_vbox_new(gboolean homogeneous, gint spacing);

Le paramètre homogeneous peut prendre deux valeurs :
- TRUE : tous les widgets occupent le même espace ;
- FALSE : chaque widget occupe l'espace qui lui est nécessaire.

Le paramètre spacing représentant l'espace entre les widgets.
Et l'association:
- *en début de box*

void gtk_box_pack_start(GtkBox* box, GtkWidget* child, gboolean expand, gboolean fill, guint padding);

- *en fin de box*

void gtk_box_pack_end(GtkBox* box, GtkWidget* child, gboolean expand, gboolean fill, guint padding);

Le paramètre expand détermine si le widget va occuper l'intégralité de l'espace fournit. TRUE pour oui, FALSE sinon.
Le paramètre fill va déterminer si le widget va remplir tout l'espace qui lui est fourni.
Le dernier paramètre permet d'ajouter de l'espace autour des widgets

17-Barre d outile

Le widget concernant la barre d'outils est appelé **GtkToolbar.** Il est l'heureux héritier de **GtkContainer**et ses ancêtres.
Pour l'initialisation. La syntaxe est : **GtkWidget* gtk_toolbar_new(void);**

Une fois la barre d'outils créée, nous allons ajouter les éléments de la barre. Et puisque, dans notre cas, les éléments seront des images qui viennent avec gtk+ (GtkStockItem), la syntaxe utilisée est :
GtkWidget* gtk_toolbar_insert_stock (GtkToolbar *toolbar, const gchar *stock_id, const char *tooltip_text, const char *tooltip_private_text, GtkSignalFunc callback, gpointer user_data, gint position);
toolbar est le nom de la barre d'outils (mabarredoutils dans notre cas).
stock_id est l'identifiant de l'image (GtkStockItem).
tooltip_text est le texte qui va avec l'élément.
tooltip_private_text n'est plus en actualité. On va le remplacer par NULL.
GtkSignalFunc callback le signal attribué à l'élément.
userdata est la donnée à passer à la fonction callback.
Et comme dans la majorité des applications, pour afficher seulement les icônes dans la barre d'outils et non les icônes et le texte. La syntaxe utilisée est :
GtkWidget* gtk_toolbar_set_style(GtkToolbar *toolbar , GTK_TOOLBAR_ICONS);

18-barres de défilement

Pour mettre en place des barres de défilement, on utilise le **widget GtkScrolledWindow**. Ce dernier est un héritier direct de **GtkBin**.
on passe à l'initialisation. La syntaxe est :
GtkWidget* gtk_scrolled_window_new(GtkAdjustment *hadjustment, GtkAdjustment *vadjustment);
Les paramètres **hadjustment** et **vadjustment** définissent les propriétés des barres de défilement horizontale et verticale. Pour nous, on va laisser ces paramètres à **NULL** pour que GTK puisse générer automatiquement deux GtkAdjustment par défaut.
GtkScrolledWindow comprend une barre de défilement horizontale et une barre de défilement verticale. Cependant on peut décider la barre qu'on veut afficher et celle qu'on ne veut pas. Dans ce cas, on utilise gtk_scrolled_window_set_policy (). Synatxe :
void gtk_scrolled_window_set_policy (GtkScrolledWindow *scrolled_window, GtkPolicyType hscroll_policy, GtkPolicyType vscroll_policy);
Les valeurs que peuvent prendre hscroll_policy et vscroll_policy sont:
GTK_POLICY_ALWAYS : si on veut que la barre de défilement soit toujours affichée.
GTK_POLICY_AUTOMATIC : si on veut que la barre de défilement soit affichée s'il est nécessaire.
GTK_POLICY_NEVER : si on veut que la barre de défilement ne soit jamais affichée.
Pour ajouter un composant **GtkScrolledWindow**, on va voir d'abord si l'élément lui-même a les capacités d'avoir ses propres barres de défilement ou pas. Si c'est le cas (exemple cas de GtkTextView), on peut utiliser **gtk_container_add ().** Dans le cas contraire, on utilisera **gtk_scrolled_window_add_with_viewport ().**
Synatxe :
void gtk_scrolled_window_add_with_viewport(GtkScrolledWindow *scrolled_window, GtkWidget *child);

19-LES ZONES DE SAISIES

Le widget est dit GtkEntry.
GtkWidget* gtk_entry_new(void);
RÉCUPÉRATION DE LADONNÉE SAISIE :
Une zone de saisie, son utilité c'est pouvoir récupérer ce qui est dedans, pour ce faire
G_CONST_RETURN gchar* gtk_entry_get_text(GtkEntry*entry);
La donnée est récupérée dans le gchar.
LA V ISIBILITÉ ET LE CARACtÈRE REMPLAÇANT :
void gtk_entry_set_visibility(GtkEntry*entry,gboolean visible);
si le gboolean =FALSE le texte est caché, cela servira à créer un champs mot de passe par exemple, et pour choisir le caractère qui remplacera les lettres(par défaut *) .
void gtk_entry_set_invisible_char(GtkEntry* entry, gunichar ch);

20-Les boutons Cases à cocher

Il s'agit tout simplement d'une **case à cocher**. Le widget qui concerne les cases à cocher est **GtkToggleButton.** Il est héritier direct de **GtkButton**.
Ici nous allons étudier deux sortes de cases à cocher. L'une utilise le widget **GtkCheckButton** qui est un héritier direct de **GtkToggleButton**. L'autre utilise le widget **GtkRadioButton** qui est un héritier direct de **GtkCheckButton.**
Ensuite on passe à l'initialisation :
Pour le CheckButon, voici les syntaxes :
GtkWidget* gtk_check_button_new (void);
GtkWidget* gtk_check_button_new_with_label (const gchar *label);
GtkWidget* gtk_check_button_new_with_mnemonic(const gchar *label);
Pour le bouton radio, voici les syntaxes :
GtkWidget* gtk_radio_button_new (GSList *group);
GtkWidget* gtk_radio_button_new_with_label (GSList *group, const gchar *label);
GtkWidget* gtk_radio_button_new_with_mnemonic(GSList *group, const gchar *label);

21-insérer une image

Le widget concernant les images est **GtkImage. GtkImage** est un fils de **GtkMisk,** donc il est héritier de **GtkMisc** et des ancêtres de **GtkMisc.**
Pour insérer une image, on commence par créer un pointeur vers **GtkWidget.** Après, on passe à l'initialisation. Plusieurs syntaxes existent, mais nous ce qui nous intéresse est la syntaxe suivante :
GtkWidget* gtk_image_new_from_file (const gchar *filename);

22-liste déroulante

La création d'une liste déroulante se fait par la fonction :
GtkWidget* gtk_combo_box_new_text();

L'ajout d'élément dans la liste se fait facilement, grâce à la fonction :
Void gtk_combo_box_append_text(GtkComboBox *combo_box,const gchar* text);
Le paramètre text étant la valeur que l'on veut insérer dans la liste.

23-Calendrier

Le widget **gtk.Calendar** est un moyen efficace pour afficher et récupérer des informations calendaires organisées par mois. Il s'agit d'un widget très simple à créer et à utiliser. Pour créer un calendrier, il suffit d'utiliser la fonction suivante :**gtk.Calendar()**
Par défaut, le calendrier affiche le mois et l'année courants.
On pourra parfois avoir besoin de modifier plusieurs données de ce widget ; les méthodes qui suivent permettront d'apporter un grand nombre de modifications au calendrier, ceci sans que l'utilisateur n'ait à voir son affichage s'actualiser à chacune d'elles
calendrier.freeze()
calendrier.thaw()

IV-Création de Macros

- ***Image :***

La fonction creerImage permet la création 'une image dans notre fenêtre.

```
GtkWidget*    creerImage(xmlNode* imageNode)
{
   GtkWidget *image;//widget image
   const xmlChar *path;//chemin de l'image
   int  width, height;//position de l'image
   // int image_type; // CH_IMG ou Q_IMG

// faire relier les paramaitres de l'image avec le fichier XML
   path = xmlGetProp(imageNode,"path");
   width= atoi(xmlGetProp(imageNode,"width"));
   height = atoi(xmlGetProp(imageNode,"height"));
// condition sur notre image
   if((width <= 0) || (width > 5000))
```

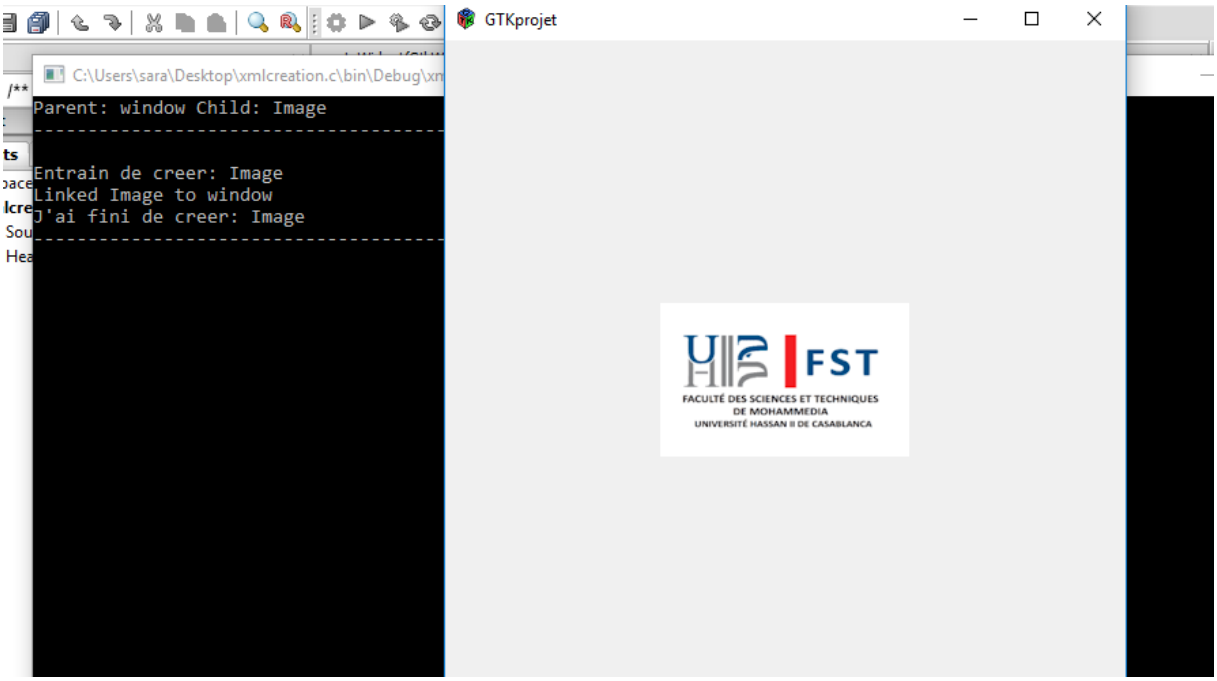
```
    width = 100;


  if((height< 0) || (height > 5000))
    height = 100;
// on associe le paramètre image a la fonction responsable de l'insertion d'une image en GTK
  image = gtk_image_new_from_file(path);
// la fonction responsable sur la position de notre image
  gtk_widget_set_size_request(GTK_WIDGET(image),width,height);


  return image;
}
```

Et pour créer une image dans un button on peut utiliser la fonction suivant :

```
GtkWidget *img=NULL ;
img=gtk_image_new_from_file(bouton->designation);
            bouton->btn = gtk_button_new ();
            gtk_button_set_image (GTK_BUTTON (bouton->btn), img);
```



Dans le fichier XML, et pour la fonction créer image on a utilisé cette balise :

```
<image path="image.png" width="10" height="10" startCol="4" startRow="5" finishRow="15" finishCol="10"/>
```

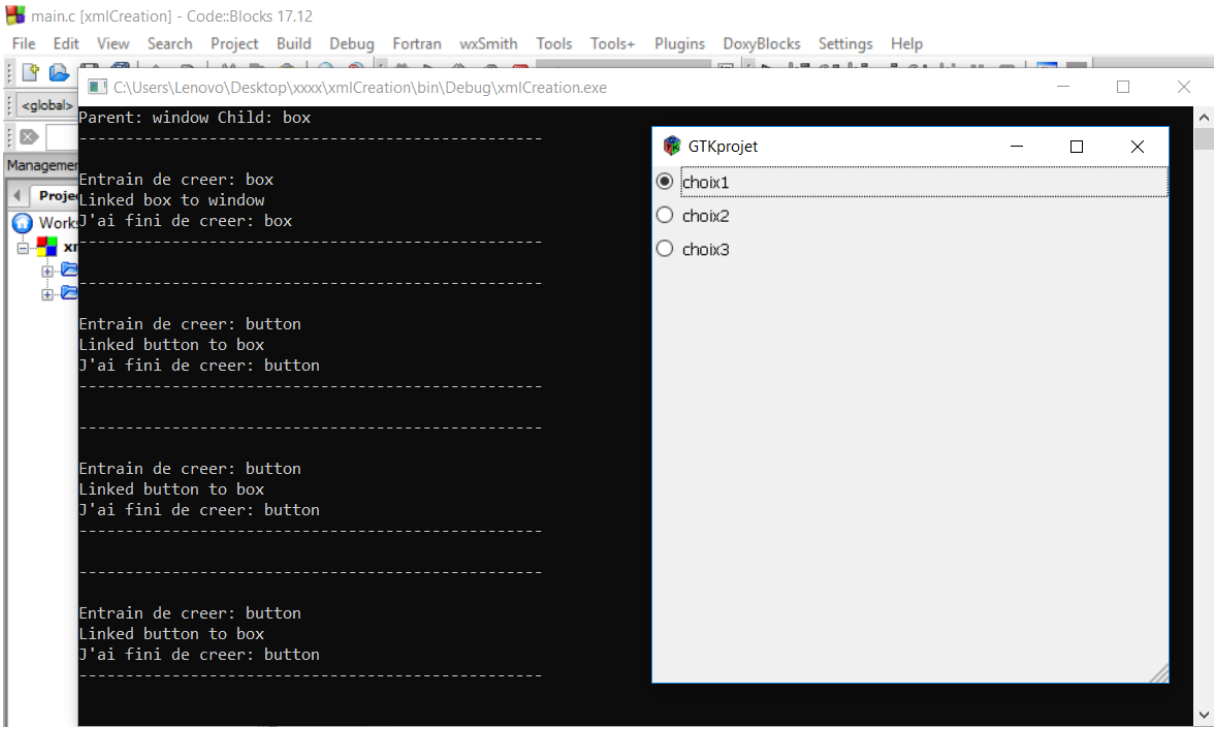
- ***Bouton Radio***

La fonction RadioBtn nous permet de créer des boutons radio sur notre fenêtre

```
GtkWidget *creerBouton(xmlNode *buttonNode)
{
  //  Variables pour stocker les propriétés du bouton
  int relief, typeBtn = 0; //déclaration des variables
  const xmlChar *label = NULL; //chemin du bouton
  //  Récupérer les valeurs des propriétés dans le fichier XML
  relief = atoi(xmlGetProp(buttonNode, "relief"));
  label = xmlGetProp(buttonNode, "name");
  typeBtn = atoi(xmlGetProp(buttonNode, "type"));
  //  Une petite vérification et condition sur notre fonction
  if ((relief < 0) || (relief > 2))
     relief = 0;
  if (!label)
     strcpy(label, "pas de texte predefini");

  //  Création du widget et affectation des valeurs récupérées
  GtkWidget *button;
  if (typeBtn == 1) //? Si le type égale à 1 on prend un bouton Radio.
  {
   //on affecte à la variable button la fonction qui crée un bouton radio dans GTK
     button = creerRadioBtn(button, label);
     return button;
  }
   //on affecte à la variable button la fonction qui crée un nouveau bouton dans GTK
  button = gtk_button_new();
```

```
   gtk_button_set_relief(button, relief);
   gtk_button_set_label(button, label);
   return button;
}
GtkWidget *creerRadioBtn(GtkWidget *button, const xmlChar *label)
{
   static GtkWidget *btnDad = NULL;
   if (btnDad == NULL)
   {
      //on affecte à la variable button la fonction qui crée un bouton radio dans GTK avec le label
       button = gtk_radio_button_new_with_label(NULL, label);
       btnDad = button; //on affecte au btnDad le paramètre button qui reçoit d'avant la fonction
   }
   else
   {
      button = gtk_radio_button_new_with_label_from_widget(btnDad, label);
   }
   return button;
```



Et pour le fichier XML, ce qui concerne la fonction bouton radio :

```
<button nature="0" type="1" name="male" event="1"></button>
                    <button nature="0" type="1" name="choix1" event="2"></button>
                    <button nature="0" type="1" name="choix2" event="3"></button>
                    <button nature="0" type="1" name="choix3" event="4"></button>
```

- ***Zone de texte :***

```
GtkWidget *createZone(xmlNode* mazone)
{
   GtkWidget *mazonedesaisie;
   GtkWidget *montexte;
   montexte=gtk_label_new(montexte);

   montexte=atoi(xmlGetProp(mazone,"montexte"));
    mazonedesaisie= gtk_entry_new();
```

```
}
```