

# Introduction à l'EDI Code::Blocks (v10)

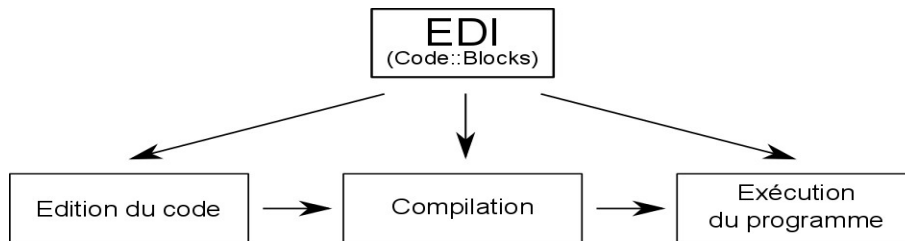
B. Baert, D. Baguette et F. Ludewig – 2013

Le logiciel Code::Blocks peut être téléchargé à l'adresse suivante :  
<http://www.codeblocks.org/downloads/binaries>

## L'EDI Code::Blocks : un Environnement de Développement Intégré (IDE - *Integrated Development Environment* en anglais)

Un environnement de développement intégré comme Code::Blocks est un programme qui regroupe un ensemble d'outils pour la programmation et le développement de logiciels. Il est principalement constitué :

- d'un éditeur de texte (pour modifier le code source) ;
- d'une interface avec un compilateur (pour générer le programme exécutable) ;
- d'un débogueur (pour analyser les erreurs dans un programme) ;
- de divers outils d'aide à la génération de code.



**Fig. 1** : Environnement de développement intégré (EDI)

### L'éditeur de code

Il s'agit d'un éditeur de texte adapté au traitement d'un code source. Celui-ci intègre généralement une coloration syntaxique automatique, qui met en évidence les mots clés du langage concerné. Lors d'un retour à la ligne, l'éditeur propose une indentation automatique du code. De plus, les numéros de ligne sont affichés dans la marge, ce qui permet de repérer plus facilement une ligne de code précise.

### Le compilateur

Le code source produit dans l'éditeur de code peut être directement compilé à partir de l'EDI : celui-ci va faire appel à un compilateur externe<sup>1</sup> qui génèrera un programme exécutable. Une fenêtre spécifique de l'EDI permet de suivre le processus de compilation et de voir les éventuels messages d'erreurs produits par le compilateur. Si tout s'est bien déroulé, l'EDI peut alors exécuter le programme compilé.

### Le débogueur

Le débogueur permet de rechercher des erreurs dans le fonctionnement d'un programme. Lors du débogage, l'EDI contrôle l'exécution du programme étape par étape et peut l'interrompre à des moments précis. Il permet également de suivre l'évolution des valeurs des différentes variables du programme tout au long de l'exécution.

---

1 Dans le cas de Code::Blocks, il s'agit de MinGW, une version adaptée à Windows du célèbre compilateur GCC

## Création d'un nouveau projet console C++ avec Code::Blocks

Lancez l'environnement de développement intégré (EDI) Code::Blocks. Créez un nouveau projet soit via le menu *File* → *New* → *Project...*, soit en cliquant sur le raccourci *Create a new project* de la fenêtre d'accueil (Fig. 2).

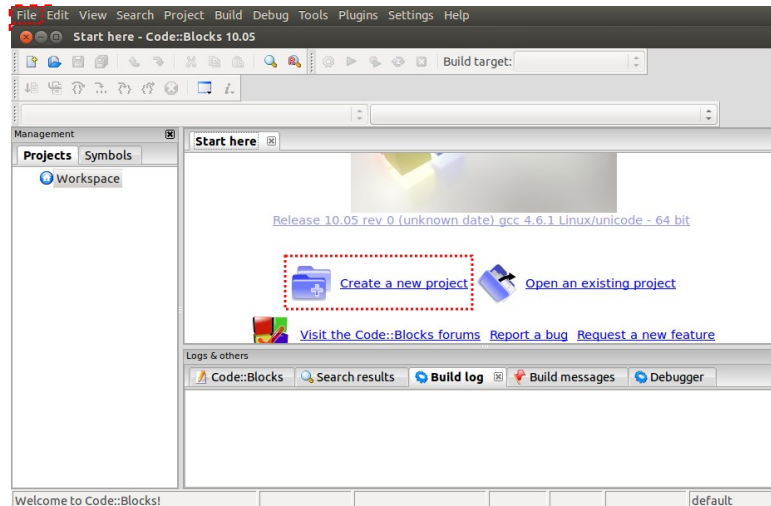


Fig. 2 : écran d'accueil

Dans la fenêtre qui s'ouvre (Fig. 3) choisissez *Projects* dans la liste de gauche et *Console application* dans la liste de droite et cliquez sur *Go*.

Remarque : Il existe une multitude de projets prédéfinis (dont plusieurs nécessitent des composants additionnels), *AVR Project* : projet pour micro-contrôleur *Atmel*, *GTK+ project* : projet avec interface graphique utilisant la bibliothèque *GTK* (multiplate-forme), *GLUT project* : projet avec gestion d'un environnement graphique 3D ...

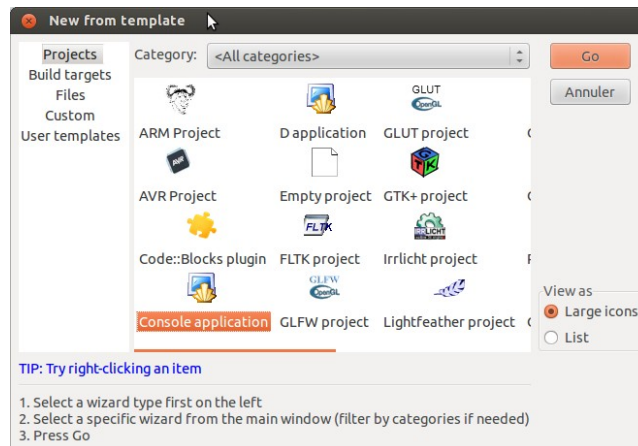


Fig. 3 : nouveau projet

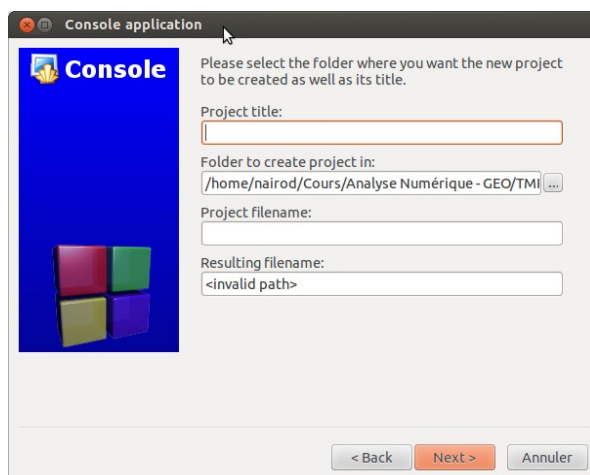
L'écran suivant permet de choisir le langage désiré. Sélectionnez *C++* et cliquez sur *Next*. La fenêtre qui suit (Fig. 4) permet de donner un nom au projet et de préciser où celui-ci doit être enregistré sur le disque.

Remarque : les projets C ou C++ sont souvent constitués de plusieurs fichiers sources (d'extension *c* ou *cpp* respectivement) et de fichiers d'en-tête (d'extension *.h*). Tous ces fichiers sont enregistrés dans un dossier portant le nom du projet donné dans *Project title* et localisés à l'emplacement spécifié par *Folder to create project in*<sup>2</sup>. En plus du dossier projet, il existe un fichier projet (d'extension *.cbp*) permettant de stocker tous les

2 Si un fichier déjà existant doit être ajouté au projet, il est préférable de copier ce fichier dans le dossier du projet.

paramètres du projet ainsi que la liste des fichiers appartenant au projet. Le nom du fichier projet (*Project filename*) est généralement le même que celui du projet. La boîte de texte *Resulting filename* donne le chemin absolu complet vers le fichier projet !

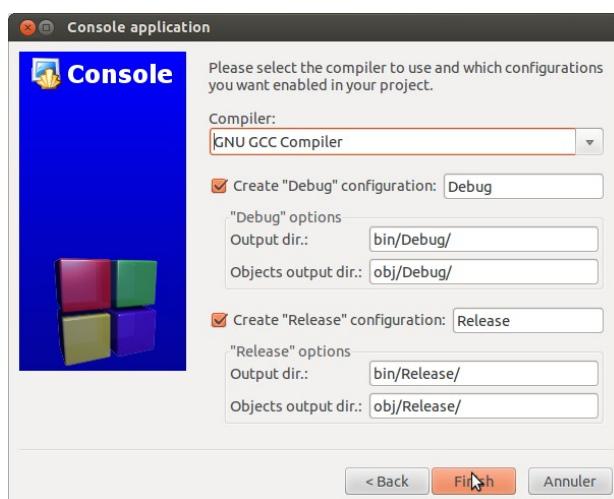
Remplissez la case *Project title* avec un nom de projet et la case *Folder to create project in* avec un chemin vers un dossier du disque dur. Laissez les deux autres cases avec leur valeur par défaut et cliquez sur *Next*.



**Fig. 4** : nom et localisation du projet

Ne modifiez rien dans la fenêtre de configuration (Fig. 5) et cliquez sur *Finish*.

Remarque : il peut exister plusieurs configurations pour un projet (ensemble de paramètres de compilation), elles sont généralement au nombre de deux. Un projet fini est généralement compilé avec des paramètres permettant de l'optimiser (vitesse, taille, ...) et porte le nom de *Release*. Un projet en cours de développement est compilé avec des informations supplémentaires afin de "surveiller" l'exécution du programme et permettre l'identification d'éventuelles erreurs. Cette configuration porte le nom de *Debug*.



**Fig. 5** : configuration du projet

Par défaut, Code::Blocks a créé un fichier nommé *main.cpp* contenant une fonction *main* et un code affichant « *Hello World !* » Vous pouvez commencer à coder directement dans ce fichier<sup>3</sup>.

---

3 Il est usuel que le fichier contenant la fonction *main* s'appelle *main.cpp*, mais ce n'est pas obligatoire.

## Gestion des fichiers dans un projet

La liste des fichiers appartenant à un projet est disponible sous le nom du projet de l'onglet *Projects* de la boîte *Management* (Fig. 6). Si la boîte *Management* n'est pas visible, affichez-la via le menu *View* → *Manager*. Il est utile de garder cette boîte à portée de main, elle est ancrable sur les bords de la fenêtre (cliquez sur la barre de titre et maintenez le bouton enfoncé, déplacez la boîte jusqu'à un rebord (un cadre semi-transparent confirme l'attache), relâchez la souris). Pour ouvrir un fichier, double-cliquez sur son nom dans la boîte *Management*<sup>4</sup>.

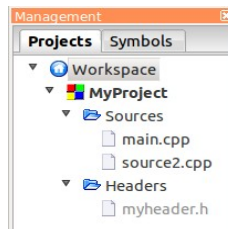


Fig. 6 : manager de projet

La liste des fichiers ouverts actuellement dans l'éditeur se situe dans la barre d'onglets de la fenêtre de travail (Fig. 7). Les fichiers précédés d'une astérisque ont subi des modifications et n'ont pas encore été enregistrés.

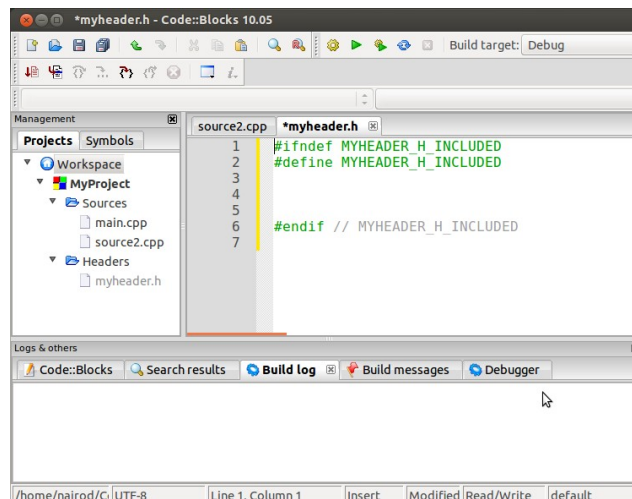

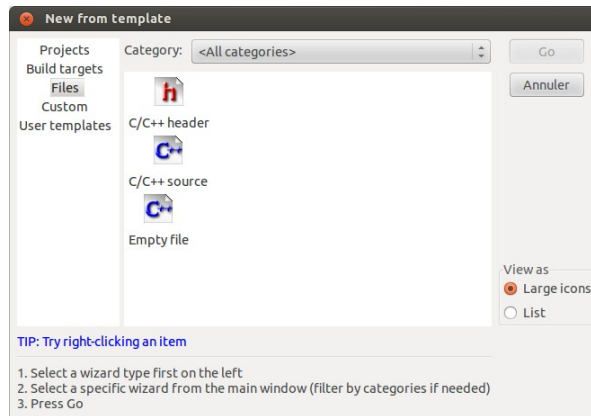


Fig. 7 : fenêtre de travail

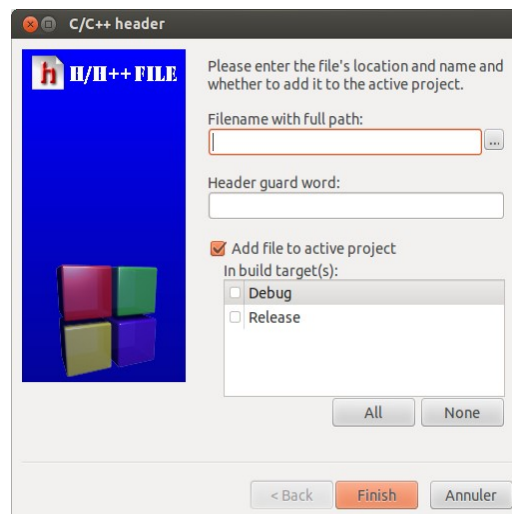
Pour ajouter un nouveau fichier, choisissez le menu *File* → *New* → *Files...* ou cliquez sur l'icône  dans la barre d'outils et choisissez *File...* (si cette barre n'est pas visible vous pouvez l'afficher via le menu *View* → *Toolbars* → *Main*). Dans la fenêtre qui s'ouvre, choisissez le type de fichier adapté dans la liste de droite (Fig. 8) et cliquez sur *Go*.

<sup>4</sup> Les fichiers présents dans le projet sont directement triés suivant leur type (source, en-tête) dans deux « dossiers » virtuels *Sources* et *Headers*. Ces dossiers n'existent pas réellement sur le disque, mais permettent de distinguer facilement les fichiers de source et les fichiers d'en-tête dans l'interface de Code::Blocks. Il est possible de rajouter d'autres dossiers virtuels afin d'organiser de gros projets.



**Fig. 8** : ajouter un nouveau fichier

La fenêtre qui s'ouvre (Fig. 9) permet de donner un nom au fichier, de préciser s'il fait partie du projet et s'il doit être compilé. Le nom du fichier doit être mis dans la case *Filename with full path*. Celle-ci doit contenir le chemin complet du fichier sur le disque (avec l'extension !), le bouton situé sur la droite de la case de texte vous localisera directement dans le dossier du projet. Cliquez sur le bouton *All*<sup>5</sup>, au bas de la fenêtre, afin d'ajouter ce nouveau fichier à toutes les configurations possibles (ceci assure que le fichier sera bien compilé). Dans le cas d'un fichier d'en-tête, une case supplémentaire *Header guard word* est présente, laissez sa valeur par défaut<sup>6</sup>.

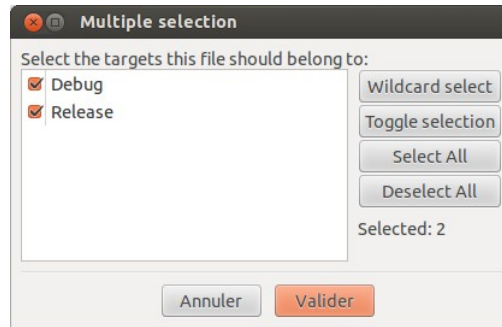


**Fig. 9** : ajouter un nouveau fichier (suite)

Pour ajouter un fichier déjà existant aux projets, choisissez dans le menu *Project* → *Add Files*. Dans la fenêtre qui s'ouvre, voyagez sur le disque jusqu'au fichier à inclure (il est préférable que ce fichier se situe dans le dossier projet) et cliquez sur ouvrir. Dans la fenêtre qui s'ouvre (Fig. 10), si le fichier doit être compilé, vérifiez que toutes les cases sont bien cochées (*Debug*, *Release*), sinon cliquez sur *Select All*. Cliquez ensuite sur *Valider/OK*.

5 Dans le cas des fichiers d'en-tête ne contenant que des déclarations, il n'est pas nécessaire (mais ce n'est pas une erreur) de rajouter ces fichiers aux différentes configurations. Les fichiers d'en-tête ne contiennent *généralement* pas de code (c'est une mauvaise habitude que d'inclure du code dans les fichiers d'en-tête) mais des déclarations, qui sont incluses dans d'autres fichiers.

6 Les fichiers d'en-tête étant inclus dans les fichiers sources (*.cpp*), il arrive que deux fichiers sources différents incluent la même en-tête (certains fichiers d'en-tête en incluent d'autres) ou des fichiers d'en-tête incompatibles et provoquent des erreurs. Pour éviter ce problème, un petit code est généré automatiquement par Code::Blocks au début du fichier, celui-ci associe au fichier un mot clé. Si le compilateur remarque l'inclusion de fichiers contenant le même mot clé, il ignore la seconde inclusion et évite les erreurs.



**Fig. 10** : inclure un fichier existant

On retire un fichier du projet en faisant un clic droit sur celui-ci dans la boîte *Management* et en sélectionnant *Remove file from project* dans le menu contextuel ou via la fenêtre de suppression de fichier dans le menu *Project* → *Remove files...* .

## Compiler et exécuter un projet

On peut compiler et exécuter un projet via la barre d'outil *compiler* (Fig. 11). Si celle-ci n'est pas visible, on peut l'afficher via le menu *View* → *Toolbars* → *Compiler*. L'icône en forme de roue dentée sert à compiler l'ensemble des fichiers du projet et à créer un fichier exécutable. L'icône en forme de triangle (play) vert sert à lancer le fichier exécutable préalablement créé par la compilation. L'icône suivante effectue ces deux opérations l'une à la suite de l'autre. Les deux flèches bleues formant un cercle servent à reconstruire l'ensemble du projet<sup>7</sup>. L'icône en forme de croix grisée devient rouge lorsque le projet est en cours de compilation ou d'exécution et permet un arrêt d'urgence. La liste déroulante *Build target* permet de déterminer la configuration à utiliser pour la compilation. Laissez celle-ci sur *Debug*<sup>8</sup> lors de l'élaboration du projet.



**Fig. 11** : barre d'outil *compiler*

Pendant la compilation, l'EDI affiche un certain nombre d'informations sur l'état d'avancement de la construction du projet dans la fenêtre de sortie (Fig. 12). Si celle-ci n'est pas visible affichez-la via le menu *View* → *Logs*. Une fois la compilation finie, si celle-ci indique *0 error(s), 0 warning(s)*, la compilation s'est bien déroulée (si seuls des *warning* sont présents, un fichier exécutable a quand-même été généré). Si des erreurs se sont produites lors de la compilation, elles sont répertoriées sous l'onglet *Build messages*<sup>9</sup>.

<sup>7</sup> L'opération de compilation étant un processus pouvant devenir long pour de gros projets, l'EDI ne recompile pas l'entièreté du projet à chaque fois, mais uniquement ce qui est nécessaire. Il peut arriver cependant, après un certain temps ou lorsque des fichiers ont été modifiés en-dehors de l'éditeur, que Code::Blocks s'y perde et ne compile pas correctement le projet. Reconstruire complètement le projet permet alors de corriger les erreurs.

<sup>8</sup> La configuration *Debug* produit un programme plus lent que la configuration *Release*, mais elle permet de détecter certaines erreurs logiques. La configuration *Release* est utilisée une fois le programme fini, la compilation est plus lente en mode *Release* car un certain nombre d'optimisations sont appliquées. Les exécutables produits par ces deux configurations différentes sont stockés dans des sous-dossiers différents du dossier projet.

<sup>9</sup> Les erreurs apparaissent les unes à la suite des autres. Lors de la correction des erreurs, il est préférable de commencer par la première car certaines erreurs peuvent en entraîner d'autres.

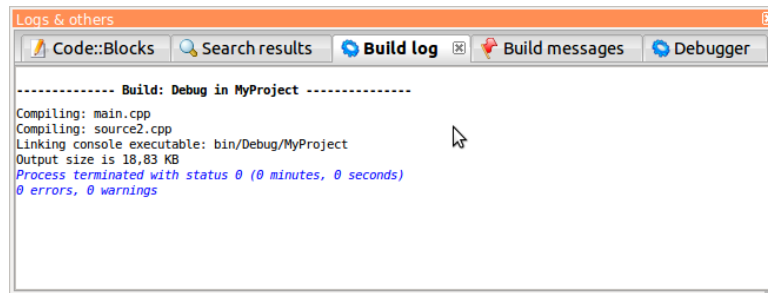


Fig. 12 : fenêtre de sortie : Build Log

## Ouvrir/Sauver/Fermer un projet

Pour ouvrir un projet préalablement enregistré, choisissez le menu *File* → *Open...* et naviguez jusqu'à votre fichier projet (extension `.cbp`). Code::Blocks peut ouvrir plusieurs projets en même temps, mais c'est rarement une bonne idée. On peut fermer un projet via le menu *File* → *Close Project*. Dans le menu *File*, vous trouverez également *Save project* et *Save all files* permettant respectivement de sauver le projet en cours et de sauver l'ensemble des modifications apportées aux fichiers. L'ensemble des fichiers d'un projet sont automatiquement sauvegardés lorsque le projet est compilé<sup>10</sup>.

## Indentation automatique dans Code::Blocks

Si votre code est mal indenté<sup>11</sup>, il existe un *plugin* par défaut dans Code::Blocks permettant de ré-indenter votre code de façon automatique. Dans le menu *Plugins* sélectionnez *Source code formatter*.

Remarque : le plugin est paramétrable suivant le style d'indentation que vous préférez. Sélectionnez *Settings* → *Editor...* dans la liste de droite de la fenêtre qui vient de s'ouvrir sélectionnez *Source formatter*. L'onglet *Style* de la partie droite permet de choisir un style prédéfini. Si vous sélectionnez *Custom* les 2 onglets *Indentation* et *Formatting* permettent de spécifier le style de mise en forme désiré.

## Déboguer un projet

L'EDI Code::Blocks permet de déboguer le code directement depuis l'éditeur. Le programme à déboguer s'exécute étape par étape. En parallèle Code::Blocks indique quelle ligne de code est actuellement exécutée. À chaque instant il est possible de connaître la valeur de toutes les variables locales, globales et les paramètres des fonctions. Cette méthode permet généralement d'identifier un code fautif mais peut s'avérer longue à mettre en œuvre.

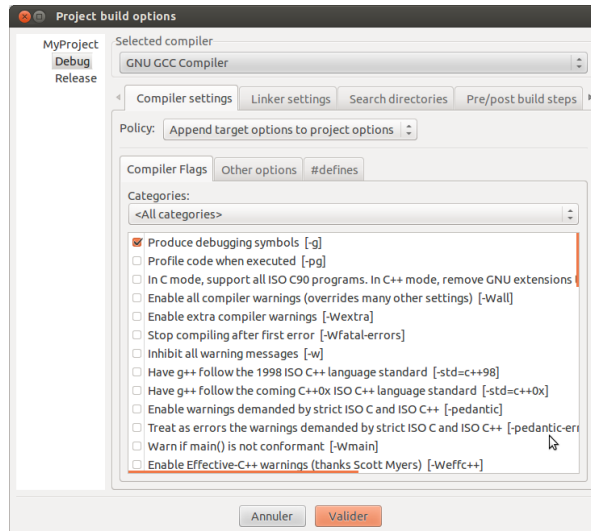
**Attention** : sur certains systèmes, le débogueur a des problèmes si le chemin vers le projet contient des espaces ou des majuscules et un message "*Error in re-setting breakpoint*" est affiché dans la fenêtre de sortie. Il arrive parfois qu'une phrase d'alerte "*warning: GDB: Failed to set controlling terminal*" soit affichée dans le terminal au début de l'opération de débogage mais ceci n'empêche toutefois pas de pouvoir déboguer correctement.

Pour profiter du suivi du code, il est nécessaire de demander au compilateur d'inclure, lors de la compilation, les informations de débogage (c'est normalement automatique pour la configuration *Debug*). Dans le menu *Project* sélectionnez *Build options...* (Fig. 13). Dans la liste de gauche vous retrouvez : le nom de votre projet, et les différentes configuration (*Debug*, *Release*) et dans la liste de droite une série d'options de compilation.

<sup>10</sup> La compilation étant effectuée par un programme différent (le compilateur), les fichiers doivent être sauvés sur le disque avant chaque compilation (ceci est fait automatiquement), sinon les modifications dans l'EDI ne seraient pas appliquées dans l'exécutable nouvellement compilé.

<sup>11</sup> Un code bien indenté est plus facile à lire. Le formatage de la source suit la structure du code et permet d'identifier rapidement le début et la fin des blocs de codes.





**Fig. 13 :** fenêtre *Build options...*

Si le nom de votre projet est sélectionné, toutes les modifications d'options que vous effectuez se répercutent sur l'ensemble des configurations. Si vous sélectionnez le mode *Release*, les modifications apportées dans les paramètres ne seront appliquées qu'en mode *Release* lors de la compilation (idem pour le mode *Debug*). Sélectionnez le mode *Debug* et vérifiez que *Produce debugging symbols* est coché (ce qui devrait normalement être le cas)<sup>12</sup>. Dans la fenêtre principale, vérifiez dans la barre de compilation (Fig. 14) que la configuration *Debug* est bien sélectionnée. Si la barre de compilation n'est pas présente vous pouvez l'afficher via le menu *View* → *Toolbars* → *Compiler*.



**Fig. 14 :** barre de compilation

L'exécution d'un programme en mode débogage ne se fait pas via la barre de compilation mais via la barre de débogage (Fig. 15). Si celle-ci n'est pas visible vous pouvez l'afficher via le menu *View* → *Toolbars* → *Debugger*.



**Fig. 15 :** barre de débogage

Il est rare (mais pas impossible) de devoir exécuter tout le programme en mode pas à pas<sup>13</sup>. On préfère généralement s'arrêter à des points clés du programme que l'on appelle points d'arrêt (*breakpoints*). On rajoute un point d'arrêt en cliquant dans la gouttière de l'éditeur (Fig. 16) qui se situe entre les numéros de lignes et l'espace d'édition. Un gros point rouge signale l'ajout d'un point d'arrêt.

<sup>12</sup> L'ajout de ces symboles de débogage alourdit considérablement l'exécutable final (un petit projet peut générer un programme exécutable de plusieurs Mo) et le temps d'exécution. Ces symboles permettent en fait au *debugger* de relier les instructions aux lignes de codes des fichiers-sources. Il existe un grand nombre d'outils permettant le débogage de code. Le profilage détermine le temps nécessaire à l'exécution de chaque partie de code (permet d'optimiser les parties les plus lentes du programme). L'analyse de fuite mémoire, .... Tous ces outils sont expliqués dans l'aide en ligne de Code::Blocks.

<sup>13</sup> On dit qu'un programme est exécuté en mode "pas à pas" quand celui-ci s'arrête après chaque instruction. Ceci laisse le temps au programmeur de vérifier le bon déroulement des choses avant d'exécuter l'instruction suivante.



```

7   int main()
8   {
9       cout.precision(20);
10      cout << fixed;
11
12      sum1 += 1.e9;
13      for(int i=0; i<1000; i++)
14          sum1 += 1.e-8;
15
16      cout << "sum1 = " << sum1;
17

```

Fig. 16 : points d'arrêt

Pour lancer l'exécution du programme jusqu'au premier point d'arrêt, cliquez sur la première icône de la barre de débogage (flèche rouge dirigée vers le bas à côté d'une page rouge). Lors de l'exécution d'un programme en mode débogage, la position actuelle du programme est affichée dans la fenêtre d'édition par une petite flèche jaune dans la gouttière (Fig. 17).







```


5   double sum1=0.,sum2=0.;
6
7   int main()
8   {
9       cout.precision(20);
10      cout << fixed;
11
12      sum1 += 1.e9;
13      for(int i=0; i<1000; i++)
14          sum1 += 1.e-8;
15
16      cout << "sum1 = " << sum1;
17

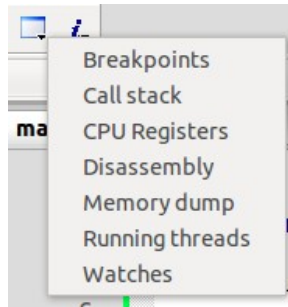
```

Fig. 17 : exécution pas à pas

Les 6 premières icônes de la barre de débogage (Fig. 15) permettent d'avancer dans le code :

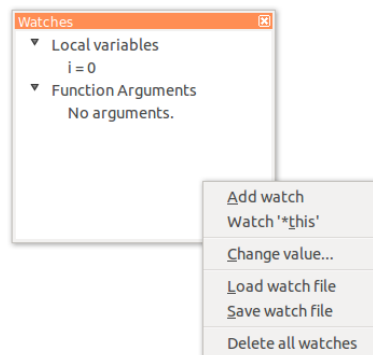
- L'icône  exécute le code jusqu'au prochain point d'arrêt
- L'icône  exécute le code jusqu'à la position du curseur
- L'icône  passe à la ligne suivante
- L'icône  passe à l'instruction suivante
- L'icône  rentre dans la fonction qui va être appelée
- L'icône  permet de sortir d'une fonction entrain d'être exécutée (le code est exécuté jusqu'au *return* ou jusque l'accolade de fermeture de la fonction)

La croix rouge  permet une interruption d'urgence du débogage. Pouvoir exécuter un programme en mode pas à pas n'aurait aucun intérêt si on ne pouvait accéder à un certain nombre d'informations lors de l'exécution. L'icône décorée d'une fenêtre bleu surmontée d'une flèche vers le bas de la barre de débogage permet d'afficher une liste de fenêtres d'informations (Fig. 18). Cette liste est aussi disponible via le menu *Debug* → *Debugging windows*.



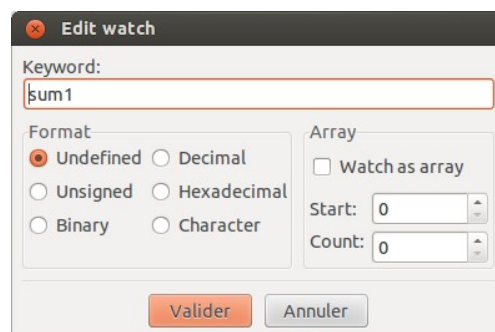
**Fig. 18** : fenêtre de débogage

La plus utile de ces fenêtres pour nous est celle nommée *Watches* (Fig. 19). Elle donne accès à l'ensemble des variables locales et aux paramètres de la fonction. Elle permet aussi de rajouter des espions. Les informations contenues dans la fenêtre sont affichées en temps réel et les variables mises à jour par la dernière instruction sont affichées en rouge.



**Fig. 19** : fenêtre *Watches* et son menu contextuel

En effectuant un clic droit dans une partie blanche de la fenêtre *Watches* vous accédez à un certain nombre de fonctions dont *Add watch* (Fig. 20). Ceci ouvre une fenêtre qui permet d'ajouter un espion sur n'importe quelle variable en ajoutant son nom dans la boîte de texte *Keyword*. Il est aussi possible d'espionner des tableaux et de préciser le type d'affichage de la variable. Une fois terminé la variable demandée est ajoutée à la fenêtre *Watches*.



**Fig. 20** : fenêtre d'édition des espions