

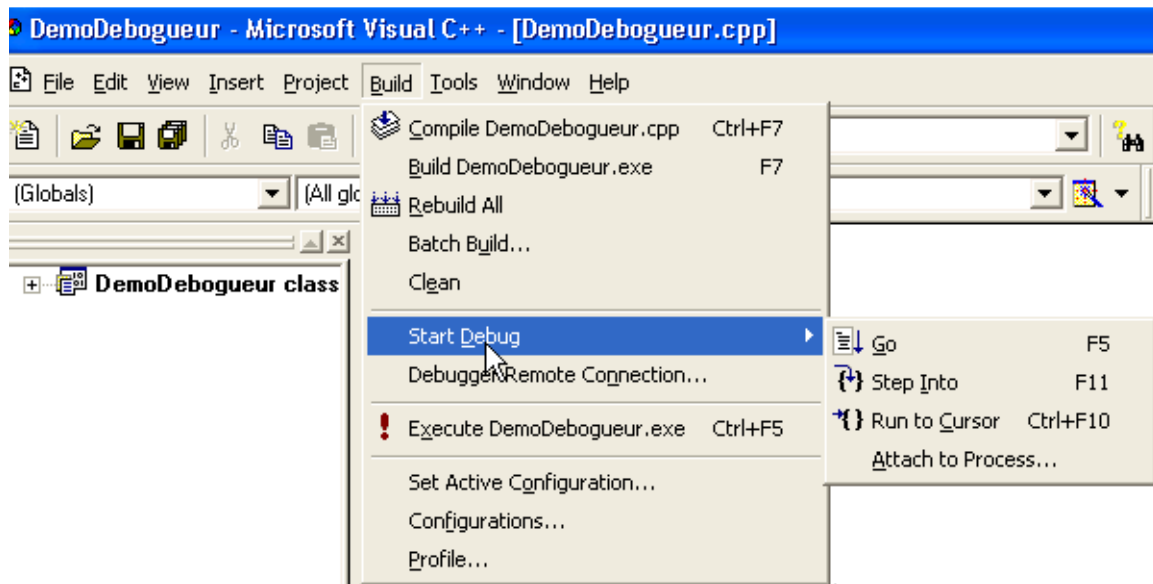
## LE DÉBOGUEUR

Pour pouvoir utiliser le débogueur, il faut tout d'abord que le programme soit compilé sans erreurs.

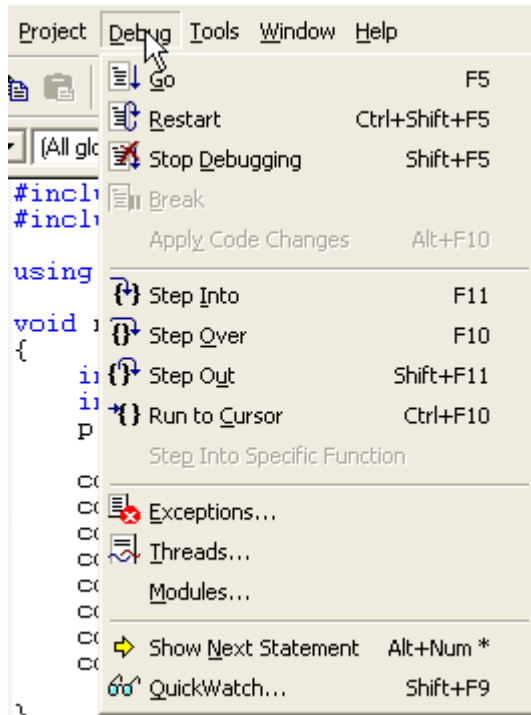
Ensuite, vous pourrez utiliser les différentes possibilités du débogueur. Voici les principales.

Lorsque le programme est compilé, on peut démarrer l'exécution du programme en mode débogage. Le menu disponible offre les choix :

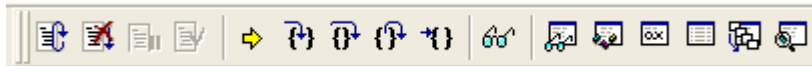
- Go (F5) : exécution en mode débogage. On utilise cette touche lorsqu'on a placé des points d'arrêt (breakpoint) dans le programme. On y reviendra.
- Step Into (F11) : Exécution instruction par instruction en entrant dans tous les sous-programmes, y compris ceux de C (voir Step over (F10) expliqué plus loin)
- Run to Cursor (Ctrl+F10) : exécution jusqu'à l'endroit où se trouve le curseur dans le programme. Pratique lorsqu'on exécute en pas à pas et qu'on veut aller plus vite en sautant une section de code qu'on sait être correcte et qui est longue à exécuter.

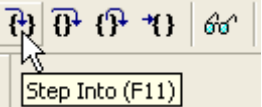
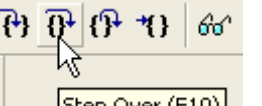
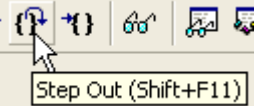
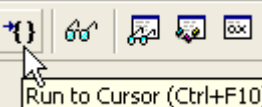


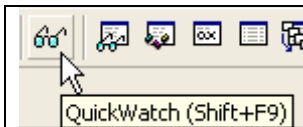
Une fois en mode débogage, un nouveau menu plus complet s'ajoute à la barre des tâches.



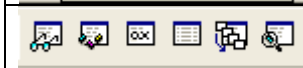
ainsi qu'une barre d'outils



 <p>Step Into (F11)</p>	<p>Exécution instruction par instruction en entrant dans tous les sous-programmes, y compris ceux de C. S'utilise en combinaison avec F10.</p>
 <p>Step Over (F10)</p>	<p>Exécution instruction par instruction. Permet de sauter le sous-programme. Le sous-programme sera exécuté mais sans qu'on le suive. Particulièrement utile pour exécuter les fonctions de C++ (cin, cout etc.) S'utilise en combinaison avec F11.</p>
 <p>Step Out (Shift+F11)</p>	<p>Exécute le programme en sortant de l'appel de fonction en cours, puis s'arrête à l'instruction qui suit l'appel de fonction. Particulièrement utile pour sortir d'une fonction de C++ dans laquelle on est entré sans le vouloir.</p>
 <p>Run to Cursor (Ctrl+F10)</p>	<p>Exécute jusqu'à l'endroit où se trouve le curseur dans le programme. Pratique lorsqu'on exécute en pas à pas et qu'on veut aller plus vite en sautant une section de code qu'on sait être correcte et qui est longue à exécuter.</p>



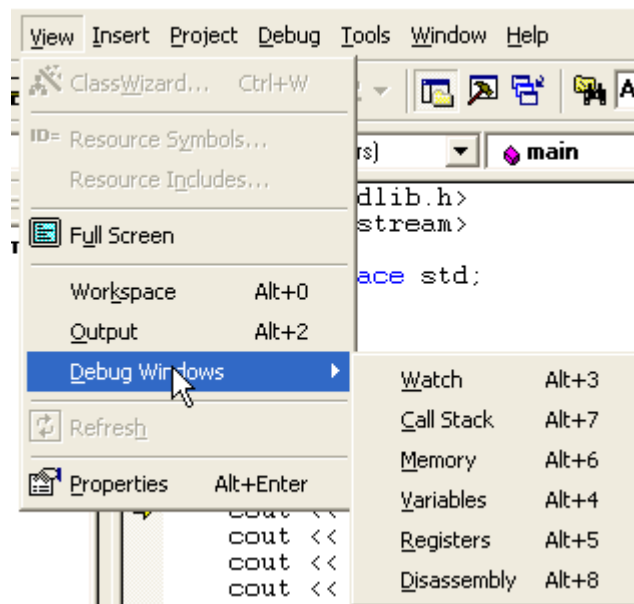
Affiche la fenêtre « Quick watch » où on peut évaluer des expression.



Quand à la dernière portion de la barre d'outils, elle sert à afficher les différentes fenêtres de débogages :

- **Watch** : On utilise la fenêtre « Watch » pour spécifier les variables et expressions qu'on veut vérifier pendant le débogage du programme. Il est aussi possible de modifier une valeur à l'aide de cette fenêtre ce qui est particulièrement utile pour vérifier le fonctionnement du programme avec certaines valeurs précises. Cette fenêtre comporte quatre onglets qui permettent de regrouper des variables à vérifier dans des contextes différents par exemple.
- **Variables** : permet un accès rapide aux variables principales dans le contexte actuel. Elle comporte trois onglets : « Auto », « Locals » et « This ». L'onglet « Auto » affiche les variables utilisées dans l'énoncé courant et le précédent. L'onglet « Locals » affiche les variables that locales à la fonction courante. Nous n'utiliserons pas l'onglet « This ».
- **Registers** : Nous ne l'utiliserons pas.
- **Memory** : On utilise la fenêtre Memory pour voir les données difficiles à voir dans les fenêtres Watch et Variables
- **Call stack** : Permet de voir le contenu de la pile des appels de fonctions.
- **Disassembly** : Nous ne l'utiliserons pas.

Ces choix sont aussi disponibles par le menu View



Pour démontrer l'utilisation du débogueur, nous utiliserons le programme suivant :

```
#include <iostream>
using namespace std;

const int MAX = 21;

void copier (char *p1, char * p2);

void main ()
{
    char chaine1 [MAX] = "";
    char chaine2 [MAX] = "";

    cin.ignore(cin.rdbuf()->in_avail());    // vider les caractères
                                           // du tampon d'entrée

    cout << "Entrez une chaine de 1 a 20 caracteres ";

    cin.getline (chaine1,20);

    cin.ignore(cin.rdbuf()->in_avail());    // vider les caractères
                                           // du tampon d'entrée

    cout << "Entrez une autre chaine de 1 a 20 caracteres ";

    cin.getline (chaine2,20);

    cout << "Chaine 1 = \" << chaine1 << "\" << endl;
    cout << "Chaine 2 = \" << chaine2 << "\" << endl;

    copier (chaine1, chaine2);

    cout << "Chaine 1 = \" << chaine1 << "\" << endl;
    cout << "Chaine 2 = \" << chaine2 << "\" << endl;

}

void copier (char *p1, char * p2)
{
    while (*p2 != '\0')    // Copier les caractères un à un
    {
        *p1 = *p2;
        ++p1;
        ++p2;
    }

    *p1 = '\0';    // Terminer la chaine copiée
}
}
```

- Copiez le programme.
- Corrigez les erreurs de syntaxe s'il y a lieu
- Démarrez le débogueur en pressant la touche F11 ou F10. Le programme commence à s'exécuter et la flèche jaune vous indique la prochaine instruction qui sera exécutée

```
#include <iostream>
using namespace std;

const int MAX = 21;


void copier (char *p1, char * p2);

void main ()
➔ {
    char chaine1 [MAX] = "";
    char chaine2 [MAX] = "";

    cin.ignore(cin.rdbuf()->in_avail());
```

- En pressant la touche F10, rendez vous à la 3e ligne

```
void main ()
{
    char chaine1 [MAX] = "";
    char chaine2 [MAX] = "";
➔    cin.ignore(cin.rdbuf()->in_avail());
```

- Si la fenêtre « Variables » n'est pas affichée, pressez sur le bouton  de la barre d'outils « Debug »
- Vous verrez chaine2 sous l'onglet « Auto »
- Basculer sous l'onglet « Local », vous verrez alors chaine1 et chaine2 qui sont les deux variables locales à « main »
- Remarquez que le + qui précède les variables permet de voir le contenu des éléments des tableaux.
- En pressant la touche F10, rendez vous au 1er appel à getline

```
➔    cin.getline (chaine1, 20);
```


- Remarquez qu'alors, le programme est bloqué même si vous appuyez sur F10 encore. C'est qu'il attend que vous entriez un chaîne dans la fenêtre d'exécution. Entrez y « Bonjour »
- Faites la même chose pour le 2e getline et entrez « Allo »
- Sur la ligne suivante, faites F11, vous verrez une fenêtre de code « bizarre » s'afficher

```

#ifdef _DLL
#pragma warning(disable:4231) /* the extern before template is a non-s
extern template class _CRTIMP basic_ostream<char, char_traits<char> >;
extern template class _CRTIMP basic_ostream<wchar_t, char_traits<wchar
#pragma warning(default:4231) /* restore previous warning */
#endif // _DLL

// INSERTERS
template<class _E, class _Tr> inline
basic_ostream<_E, _Tr>& __cdecl operator<<((
basic_ostream<_E, _Tr>& _O] const _E *_X)
⇒ | {typedef basic_ostream<_E, _Tr> _Myos;
ios_base::iostate _St = ios_base::goodbit;
size_t _N = _Tr::length(_X);
size_t _M = _O.width() <= 0 || _O.width() <= _N
? 0 : _O.width() - _N;
const _Myos::sentry _Ok(_O);
if (!_Ok)



```

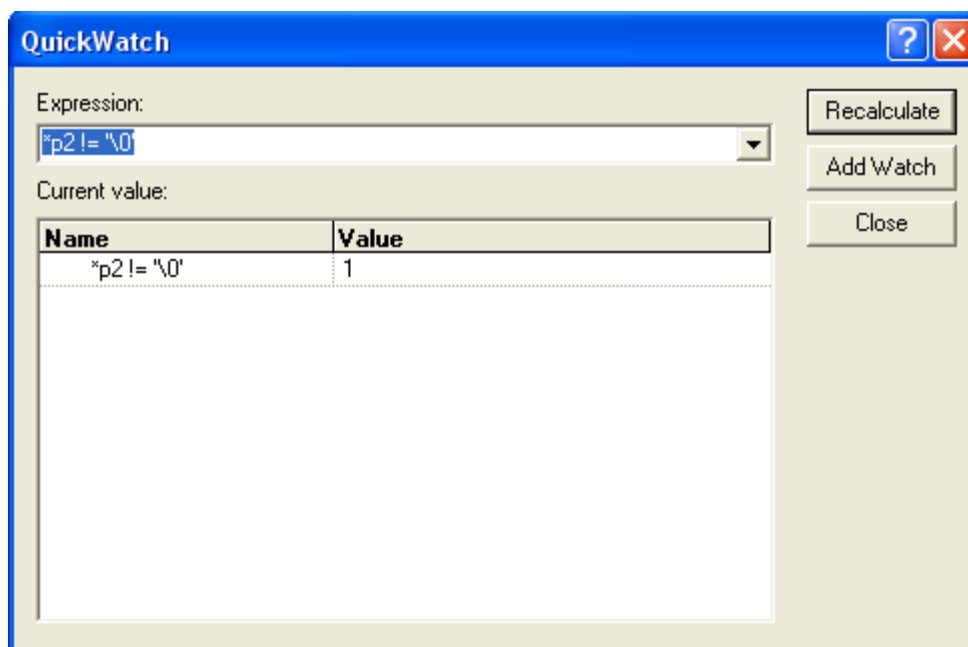
- Il s'agit du code de la méthode cout. Si ça vous arrive, vous n'avez qu'à presser le bouton Step out  de la barre d'outils pour en sortir. Faites le.
- Rendez-vous à l'appel de la procédure Copier puis faites F11 cette fois on veut vraiment entrer dans la procédure.
- Remarquez que la fenêtre des variables n'affiche plus maintenant chaine1 et chaine2 mais p1 et p2 c'est normal puisque les variables locales ne sont plus les mêmes.
- Vous pouvez continuer l'exécution en pas à pas (F10) jusqu'à la fin.

### Les points d'arrêt (breakpoint)

- Terminez l'exécution précédente.
- Placez-vous sur la ligne du while de la procédure copier et faites F9. Vous verrez apparaître un gros point rouge dans la marge gauche, vis-à-vis le while. (Si vous voulez enlever un point d'arrêt, refaites F9)
- Faites F5, le programme exécutera sans s'arrêter jusqu'à la ligne sur laquelle se trouve le point d'arrêt. Vous entrerez donc les deux chaînes puis le programme arrête et attend.
- À partir de ce moment, vous pouvez continuer ligne par ligne avec F10 ou F11 ou encore poser d'autres points d'arrêt et refaire F5 pour s'y rendre.
- À chaque fois que le débogueur arrête, vous pouvez vérifier le contenu des variables ou évaluer des expressions.

## Vérifier le contenu des variables

- On a vu qu'on peut voir le contenu de nos variables à l'aide de la fenêtre « Variables » mais parfois on a besoin d'isoler certaines variables (quand on en a beaucoup) ou bien d'évaluer une expression arithmétique ou logique. On utilise alors la fenêtre « Watch ».
- Affichez la fenêtre « Watch » à l'aide du bouton . Ajustez sa taille si nécessaire. Vous pouvez aussi la déplacer en la tenant par le cadre.
- Dans la case sous « Name » écrivez \*p1 et, dans la case en dessous \*p2
- Vous verrez leur valeur apparaître dans la colonne « value »
- Mettez en surbrillance l'expression \*p2 != '\0' dans votre code et cliquez sur le bouton  pour afficher la fenêtre QuickWatch
- Vous verrez que l'expression est affichée dans la fenêtre et sa valeur évaluée en tenant compte des données présentes. N'oubliez pas que 1 équivaut à TRUE en C.



- Cliquez sur le bouton « Add Watch », l'expression sera ajoutée à la fenêtre Watch. Notez que vous pouvez aussi copier l'expression directement dans la fenêtre Watch.
- Maintenant, si vous exécutez en pas à pas (F10 ou F11) ou par saut aux points d'arrêt (F5), vous pouvez observer les changements dans vos variables et votre expression logique.
- La fenêtre Watch vous permet aussi de changer la valeur d'une variable au cours de l'exécution. Il suffit de double-cliquer sur la valeur que vous voulez changer dans la colonne « Value »