

Travaux Pratiques n° 1

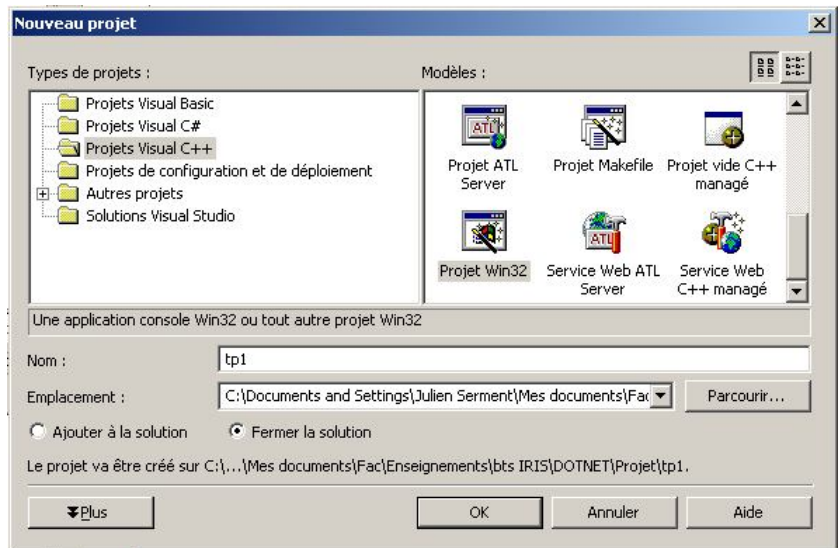
Premiers programmes

Exercice 1 : Mise en route

1. Création d'un Projet Visual C++

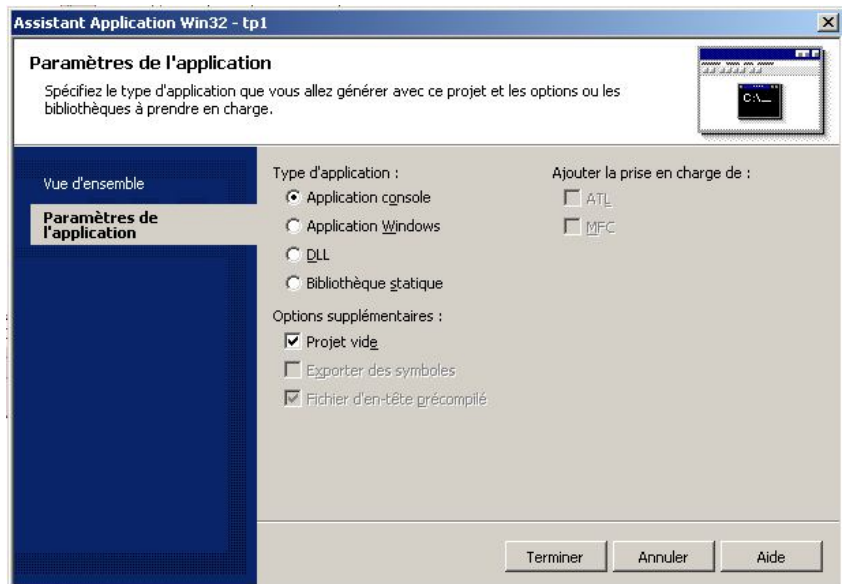
- Démarrer / Programmes / Microsoft Visual Studio .NET
- Fichier / Nouveau / Projet

- Assistant Projet
 - Projet Visual C++
 - Projet win32
 - Saisir le nom du projet
 - Saisir l'emplacement
 - Cliquer sur bouton OK



Assistant Application

- Paramètres de l'application
- **Application console**
- Projet vide



2. Création/Ajout d'un fichier .cpp

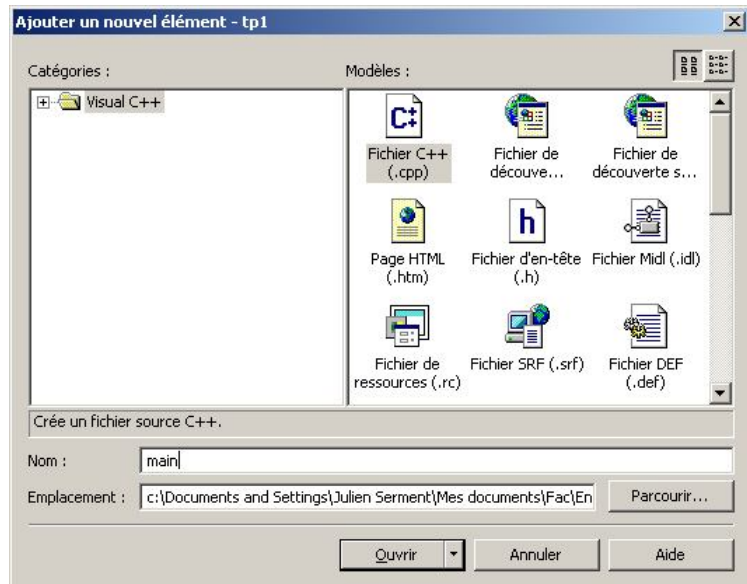
- *Projet / Ajouter un nouvel élément*

- Assistant Ajout élément

- Visual C++
- Fichier C++ (.cpp)

- Saisir le nom du fichier
- Saisir l'emplacement

- Cliquer sur bouton Ouvrir



- Saisir le code suivant dans le fichier créé :

```
#include<iostream.h> // bibliothèque nécessaire à l'exécution de l'instruction cout
```

```
int main(void) // point d' entrée du programme
```

```
{
```

```
cout << "Bonjour !" << endl ;
```

```
/* cout permet d' écrire sur la sortie standart
```

```
endl signifie un passage à la ligne*/
```

```
return 0;//car la fonction main doit retourner un entier (int)
```

```
}
```

3. Compilation et exécution du projet

Il existe également plusieurs possibilités pour l'**exécution** du projet. Nous n'en verrons qu'une pour l'instant : l'exécution sans débogage : *Débuguer / Exécuter sans débogage* (Ctrl+F5)

Exécutez votre projet. Une fenêtre DOS doit s'ouvrir et afficher « Bonjour ! ».

Exercice 2 : string et entrée / sortie

Sélectionner dans le menu *Fichier / Enregistrer main.cpp sous...* donner le nom `exo1.cpp`.

Le code du programme de l'exercice 1 est ainsi sauvegardé. Dans la fenêtre **explorateur de solutions** cliquer sur *TP1 / Source Files / main.cpp*.

Modifier le fichier `main.cpp` avec le code suivant :

```
#include<iostream>
#include<string> //bibliothèque contenant le type string
using namespace std; //Pour l'accès aux bibliothèques
int main(void)
{
    const string b("bonjour"); //Déclaration et initialisation
    string prenom; //déclaration
    prenom = "thibault"; //puis affectation
    cout << b << prenom << endl;
    return 0;
}
```

Compiler (*Générer / Compiler* ou `Ctrl+F7`) et exécuter le projet en mode sans débogage (*Déboguer / Exécuter sans débogage* ou `Ctrl+F5`).

Modifier le code afin que le prénom soit demandé à l'utilisateur et saisi.

Exercice 3 : Instructions élémentaires

Sélectionner dans le menu *Fichier / Enregistrer main.cpp sous...* donner le nom `exo2.cpp`. Le code du programme de l'exercice 2 est ainsi sauvegardé.

Modifier le fichier `main.cpp` en suivant les instructions suivantes :

Caractère et entiers :

- Déclarer 1 variable nommé `i` de type entier simple (`int`)
- Déclarer 1 constante nommée `c` de type caractère (`char`) ayant pour valeur `'\a'`
- Affecter une valeur à `i`
- Afficher `i` et `c` à l'aide de l'instruction `cout`.

Compiler (*Générer / Compiler* ou `Ctrl+F7`) et exécuter le projet sans débogage (*Déboguer / Exécuter sans débogage* ou `Ctrl+F5`).

Les entiers :

- Déclarer 2 variables `i` et `l` de type entier (`int`) et entier long (`long int`)
- Demander à l'utilisateur de saisir 2 entiers
- Affecter les valeurs saisies à `l` et `i`
- Afficher le résultat de la division de `l` par `i`.
- Afficher le reste de la division de `l` par `i`.

Compiler et exécuter le projet sans débogage.

Les réels

- Déclarer 1 variables `f` de type réel (`float`)
- Affecter une valeur à `f`
- Afficher le résultat de la division de `l` par `f`.
- Afficher le reste de la division de `f` par `i`.

Compiler et exécuter le projet sans débogage.

Les booléens

- Déclarer 3 variables b1, b2 et b3 de type booléen (bool)
- Affecter à b1 la valeur true
- Afficher la valeur de b1

Compiler et exécuter le projet sans déboguage.

- Affecter à b2 le résultat de la comparaison $i > j$
- Affecter à b3 le résultat de la conjonction ou ET logique (&&) de b1 et b2
- Afficher les valeurs de b2 et b3

Compiler et exécuter le projet sans déboguage.

Exercice 4 : Utilisation du Débogueur

Créer un nouveau projet nommé Exo4 et le fichier main.cpp contenant le code ci-dessous :

```
#include<iostream>
using namespace std;
int main(void)
{
    int i,j;
    i="33";
    cout << "somme = " << i + j << endl;
    return 0;
}
```

Que se passe-t-il lors de la compilation (*Générer / Compiler* ou Ctrl+F7) ?

L'erreur (de compilation) étant facile à détecter ici (33 au lieu de "33" car i est un entier et non une chaîne de caractère), observez bien ce qui est inscrit dans la fenêtre **Liste des tâches** afin de comprendre comment le débogueur signale les erreurs de compilation.

Double cliquez sur la ligne décrivant l'erreur et corrigez la en enlevant les guillemets.

Compilez à nouveau et exécutez le projet sans déboguage (*Déboguer / Exécuter sans déboguage* ou Ctrl+F5). Que se passe-t-il ?

Il s'agit cette fois d'une erreur d'initialisation, aucune valeur n'ayant été affectée à la variable j. Observez bien ce qui est inscrit dans la fenêtre d'affichage de déboguage afin de comprendre comment le débogueur signale les erreurs d'exécution.

Sans corriger l'erreur, exécutez le projet en mode déboguage (*Déboguer / Démarrer* ou F5). Cliquez sur le bouton **Arrêter** : vous êtes sur la ligne contenant l'instruction déclenchant l'erreur.

Sans corriger l'erreur, exécutez le projet en mode Pas à pas principal (*Déboguer / Pas à pas principal* ou F10) en observant l'évolution des valeurs des variables locales dans la fenêtre **Variables locales**.

Modifier votre code de manière à ce que votre programme calcule la somme de 5 entiers initialisés avec les valeurs de votre choix. Afin de vérifier qu'il n'y ait plus d'erreur d'initialisation, mettez un **point d'arrêt** au niveau de la ligne de l'instruction affichant le résultat de la somme à l'écran. Exécutez en mode déboguage (*Déboguer / Démarrer* ou F5), quand l'exécution se mets en pause, vérifiez dans la fenêtre **Variables locales** que les 5 variables ont bien été initialisées. Une fois que votre application ne semble plus contenir d'erreur, exécutez la sans déboguage (*Déboguer / Exécuter sans déboguage* ou Ctrl+F5).

Environnement de développement intégré

(IDE, *Integrated Development Environment*)

Visual Studio .Net

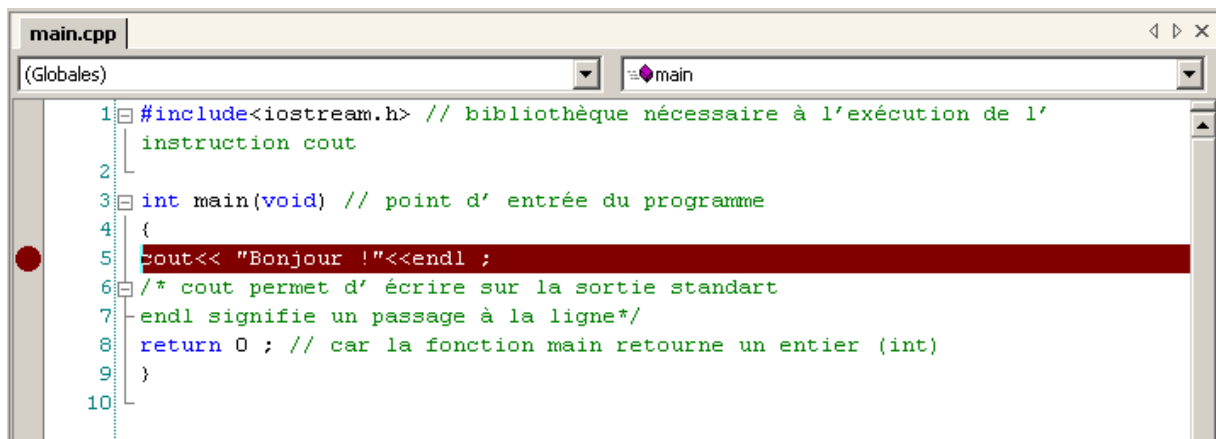
Le débogueur

Par défaut, Visual C++ compile les projets en mode "**Debug**". Cela signifie que des informations de débogage sont incluses dans l'exécutable, afin d'aider le programmeur à retrouver certaines erreurs. En effet, le programme est alors **traçable**, c'est-à-dire qu'à tout moment durant l'exécution, il est possible de savoir où on se trouve dans le code. De même, lors d'un plantage, le débogueur pourra rendre la main si on le désire, exactement à l'endroit où l'erreur a eu lieu. Ceci simplifie la tâche du développeur.

Tracer un programme, consiste à l'exécuter ligne par ligne, expression après expression. Pour ce faire, il y a plusieurs solutions.

Points d'arrêt

La première solution fonctionne suivant la technique des "**points d'arrêt**" : on place un point d'arrêt quelque part dans le code (Ctrl+B ou clique dans la marge grisée), à une ligne donnée.



```
main.cpp | (Globales) | main
1 | #include<iostream.h> // bibliothèque nécessaire à l'exécution de l'
  | instruction cout
2 |
3 | int main(void) // point d' entrée du programme
4 | {
5 | cout<< "Bonjour !" << endl ;
6 | /* cout permet d' écrire sur la sortie standart
7 | endl signifie un passage à la ligne*/
8 | return 0 ; // car la fonction main retourne un entier (int)
9 | }
10 |
```

On obtient un petit rond rouge sur la gauche de la ligne de code, qui signifie qu'un point d'arrêt est présent.

On lance alors l'exécution en mode débogage (*Déboguer / Démarrer* ou F5), et le programme s'arrête automatiquement à l'endroit voulu. Il est alors possible de le relancer jusqu'à la fin ou au prochain point d'arrêt.

Débogage Pas à Pas

L'une des procédures de débogage les plus courantes est l'exécution pas à pas : le code est exécuté ligne par ligne.

Dans le menu *Déboguer*, trois commandes permettent d'exécuter le code pas à pas :

- *Pas à pas détaillé*
- *Pas à pas principal*
- *Pas à pas sortant*

Pas à pas détaillé et *Pas à pas principal* ne diffèrent que par un aspect — leur façon de gérer les appels aux fonctions. Les deux commandes demandent au débogueur d'exécuter la ligne suivante dans le code. Si la ligne suivante contient un appel à une fonction, *Pas à pas détaillé* n'exécute que l'appel, puis s'arrête à la première ligne de code se trouvant dans la fonction. *Pas à pas principal* exécute la fonction en entier, puis s'arrête à la première ligne se trouvant en dehors de la fonction. Utilisez *Pas à pas détaillé* pour vérifier le code à l'intérieur de la fonction appelée. Utilisez le *Pas à pas principal* si vous voulez éviter d'entrer dans les fonctions.

Dans un appel à plusieurs fonctions imbriquées, *Pas à pas détaillé* va jusqu'à la fonction se trouvant au niveau le plus profond. Si vous utilisez *Pas à pas détaillé* dans un appel tel que `Func1(Func2())`, le débogueur va dans la fonction `Func2`. Pour choisir la fonction imbriquée dans laquelle le débogueur doit pénétrer, dans le menu contextuel, utilisez la commande *Pas à pas détaillé* de la fonction spécifique. Pour plus d'informations, consultez l'aide Microsoft.

Pour sortir d'une fonction imbriquée et revenir à la fonction appelante, utilisez *Pas à pas sortant*. *Pas à pas sortant* reprend l'exécution de votre code jusqu'au retour de la fonction, puis assure une interruption au point de retour dans la fonction appelante.

Les commandes Pas à pas sont inaccessibles si votre application est en cours d'exécution. Les commandes Pas à pas ne sont valides qu'en mode arrêt ou avant le démarrage de l'application. Pour plus d'informations, consultez l'aide Microsoft.

Les données

Il est très utile pour un développeur de pouvoir tracer un programme, mais il est peut-être plus intéressant encore de pouvoir connaître l'état des données (variables) du programme. Les valeurs des variables en cours d'exécution peuvent être affichées dans une fenêtre **Variables locales** (*Déboguer / Fenêtres / Variables locales*).

Si vous voulez connaître la valeur d'une variable tout au long de l'exécution de l'application, vous pouvez utiliser des « **espions** ». Pour ajouter un espion, il suffit de faire un clic droit sur la variable sélectionnée et cliquer sur *ajouter un espion*. Les différents espions peuvent être affichés ou cachés (*Déboguer / Fenêtres / Espion*). *Espion express* vous permet de connaître la valeur actuelle d'une variable. La fenêtre **Automatique** (*Déboguer / Fenêtres / Automatique*) affiche toutes les variables se trouvant dans l'instruction en cours ainsi que dans l'instruction précédente.

Signets

L'IDE de Visual Studio propose différentes méthodes pour se déplacer dans les documents pour rechercher des lignes de code.

Pour marquer et extraire des lignes de code :

- Créez des signets pour toutes les lignes de code d'un fichier, puis parcourez tous les signets en utilisant les commandes **Signet suivant / Signet précédent**.
- Ajoutez des annotations à votre code, puis cliquez sur leurs raccourcis dans la **Liste des tâches** pour les retrouver.
- Passez d'une fenêtre de document ouverte à l'autre en appuyant sur **Ctrl+Alt+Tab**.
- Déplacez-vous dans l'historique des modifications apportées à un ou plusieurs documents en cliquant sur **Précédent** et **Suivant** dans la barre d'outils standard.

Pour plus d'informations, consultez l'aide de Microsoft Visual Studio.