TP Débugueur

Sommaire

Les objectifs de ce TP sont d'être capable de débuguer un programme C/C++.

Débugueur

Présentation

Un débogueur (ou débugueur, de l'anglais *debugger*) est un logiciel qui aide un développeur à analyser les *bugs* d'un programme. Pour cela, il permet **d'exécuter le programme pas-à-pas, d'afficher la valeur des variables à tout moment, de mettre en place des points d'arrêt sur des conditions ou sur des lignes du programme** ...

Le programme à débuguer est exécuté à travers le débogueur et s'exécute normalement. Le débogueur offre alors au programmeur la possibilité d'observer et de contrôler l'exécution du programme.

Lire : fr.wikipedia.org/wiki/Débogueur

GNU Debugger

GNU Debugger, également appelé `gdb`, est le débogueur standard du projet GNU. Il est portable sur de nombreux systèmes type Unix et fonctionne pour plusieurs langages de programmation, comme le C et le C++. Il fut écrit par Richard Stallman en 1988. `gdb` est un logiciel libre, distribué sous la licence GNU GPL.

`gdb` permet de déboguer un programme en cours d'exécution (en le déroulant instruction par instruction ou en examinant et modifiant ses données), mais il permet également un débogage post-mortem en analysant un fichier `core` qui représente le contenu d'un programme terminé anormalement.

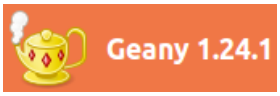
L'interface de `gdb` est une **simple ligne de commande**, mais il existe des applications qui lui offrent une interface graphique beaucoup plus conviviale (`ddd`, `nemiver`, ...). `gdb` est souvent invoqué en arrière-plan par les environnements de développement intégré (Eclipse, `geany`, ...).

Important : Les programmes C/C++ doivent être compilés avec l'option -g pour pouvoir être débuggés par gdb.

Outils

Geany est un éditeur de texte avec des fonctions basiques d'environnement de développement intégré (EDI) utilisant de nombreux *plugins*. Pour disposer de la dernière version, il vous faut faire :

```
$ sudo add-apt-repository ppa:geany-dev/ppa
$ sudo apt-get update
$ sudo apt-get install geany
```



Pour installer quelques *plugins* intéressants, il faut faire :

```
$ sudo apt-get install geany-plugin-addons geany-plugin-codenav geany-plugin-scope geany-
    plugin-debugger geany-plugin-vc
```

Nemiver est une interface graphique de `gdb` s'intégrant bien à Gnome. Pour l'installer, il faut faire :

```
$ sudo apt-get install nemiver
```


Travail demandé

Exercice n°1 : prise en main

On désire déboguer le programme suivant qui comporte trois erreurs :

```
#include <iostream>

using namespace std;

float calculTTC(float, float);

int main(void)
{
  float prix_ht, prix_ttc, tva;

  cout << "Calcul du prix TTC\n\n";

  cout << "Entrez un prix HT et la TVA : ";
  cin >> prix_ht >> tva;

  prix_ttc = calculTTC(tva, prix_ht);

  cout << "\n" << prix_ht << " euros HT = " << prix_ttc << " euros TTC\n";

  return 0;
}

float calculTTC(float prix_ht, float tva)
{
   float prix_ttc;

   prix_ttc = prix_ht * tva / (100 + prix_ht);
   //prix_ttc = (prix_ht + prix_ht * tva) / 100;

   return prix_ht;
}
```

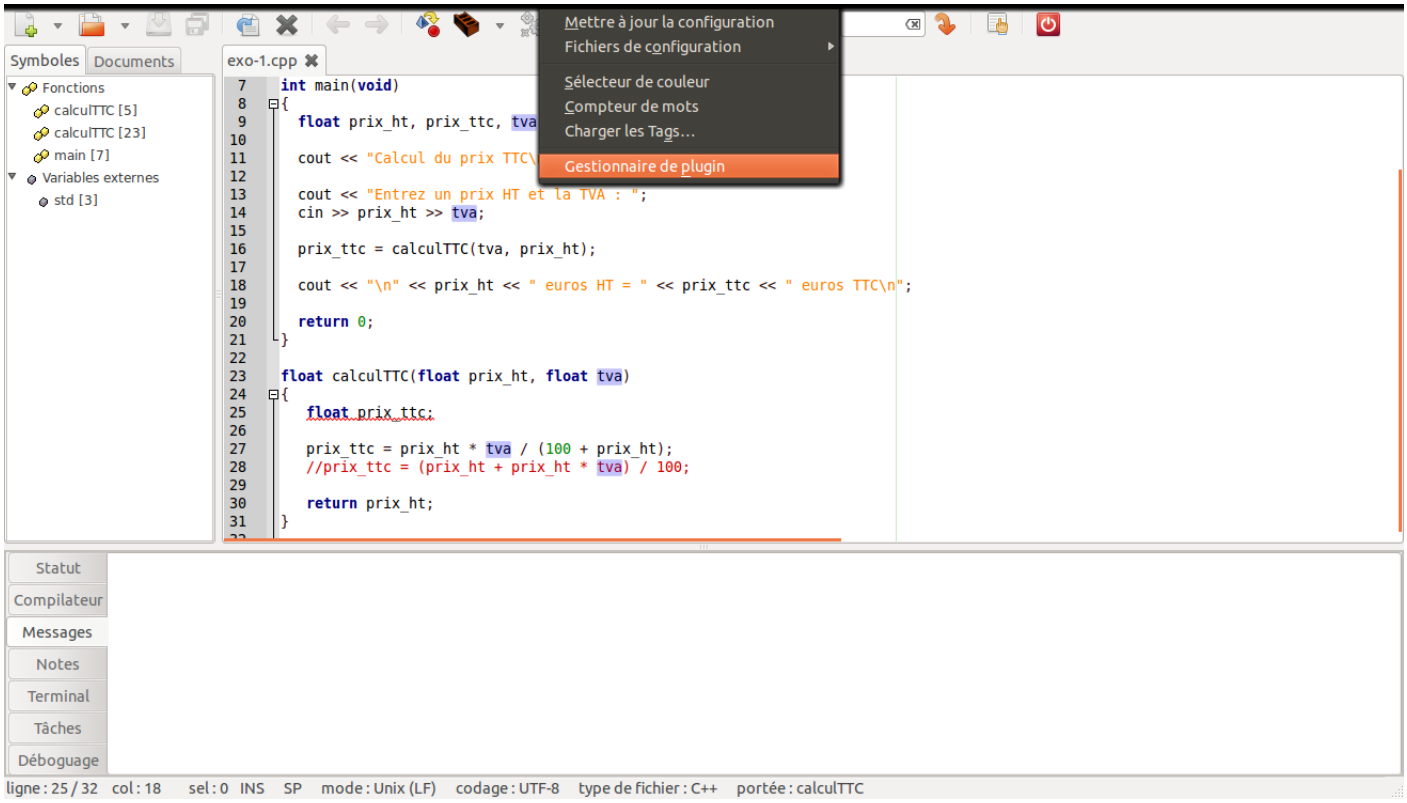
exo-1.cpp

Cette partie détaille l'utilisation de débogueurs avec **Geany** et **Nemiver**.

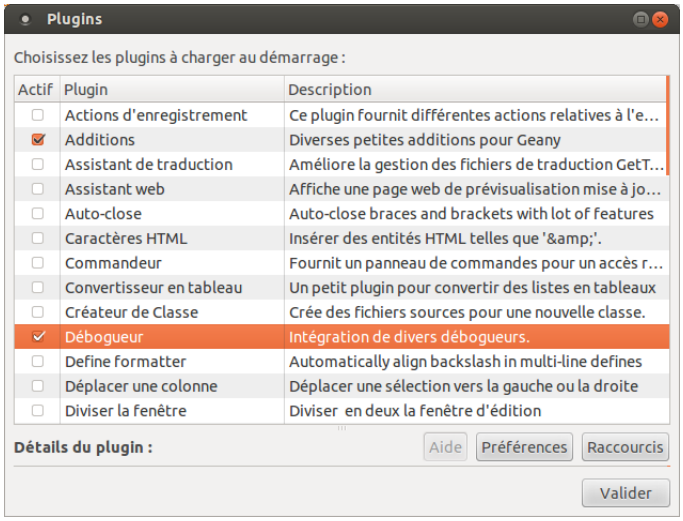
Geany propose l'intégration de deux débugueurs : `geany-plugin-debugger` et `geany-plugin-scope`.

geany-plugin-debugger

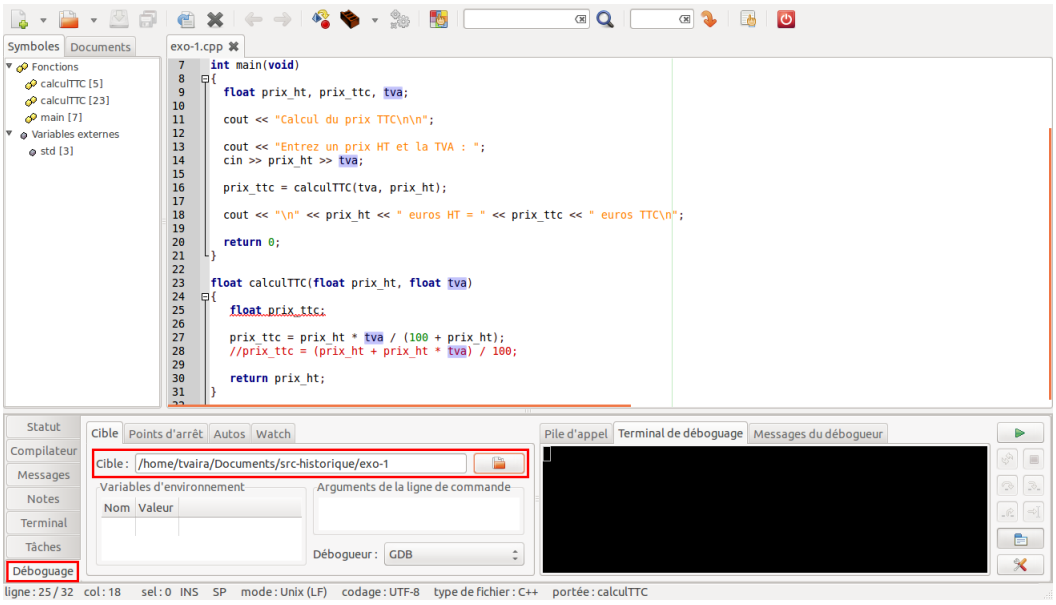
Aller dans le menu “Outils” puis dans “Gestionnaire de plugin ” :



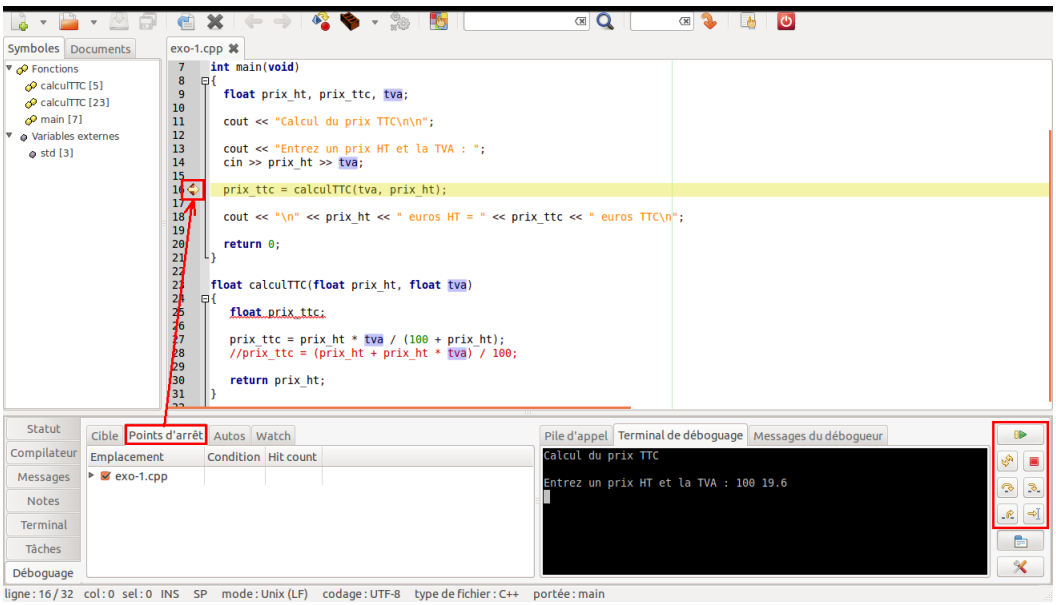
Activer le *plugin* Débogueur :



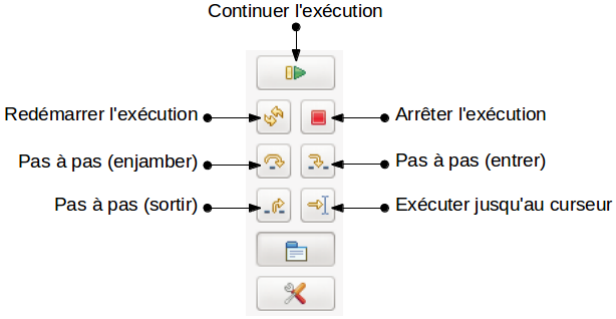
Sélectionner l'onglet "Déboguage" puis "Cible". Il vous faut choisir l'exécutable à déboguer :



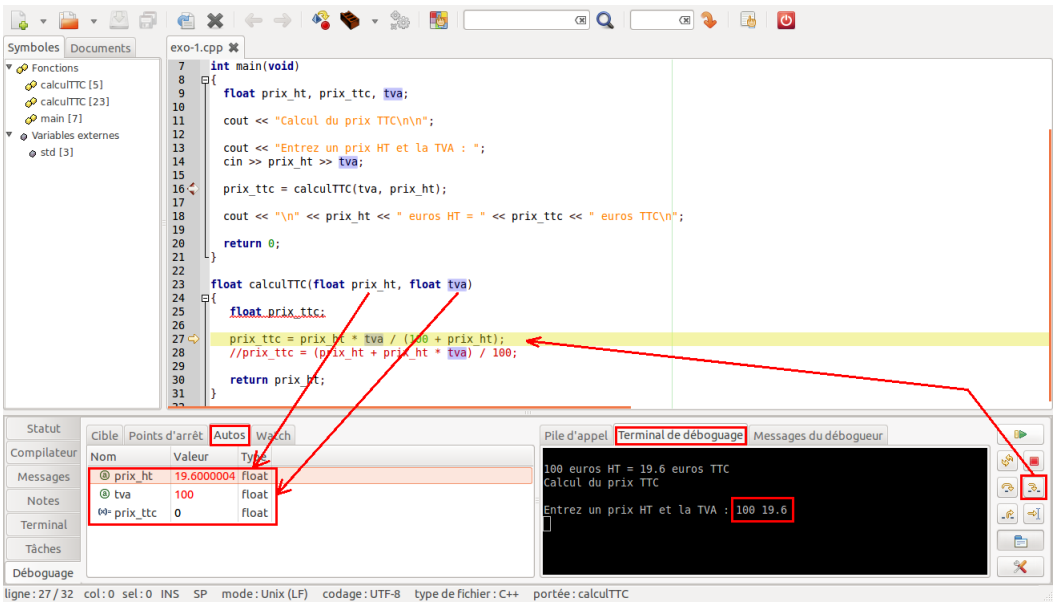
Vous pouvez placer des points d'arrêts en cliquant sur les numéros de lignes (ici la 16) puis démarrer l'exécution dans le "Terminal de déboguage" :



Le contrôle de l'exécution du programme se fait à l'aide d'icônes :

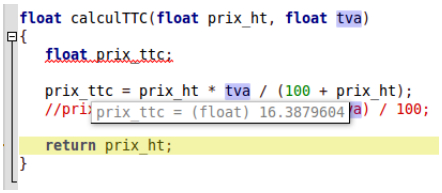


On sélectionne le mode d'exécution pas à pas (entrer *Step Into*) puis on observe les variables dans l'onglet "Autos" :



On constate que les variables `tva` et `prix_ht` ont été inversées au moment de l'appel à la fonction `calculTTC()` !

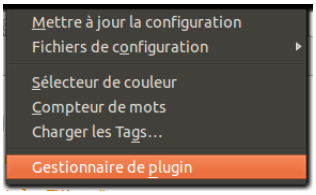
Il est aussi possible de placer le curseur de la souris sur une variable pour afficher sa valeur (ici la variable `prix_ttc`) :



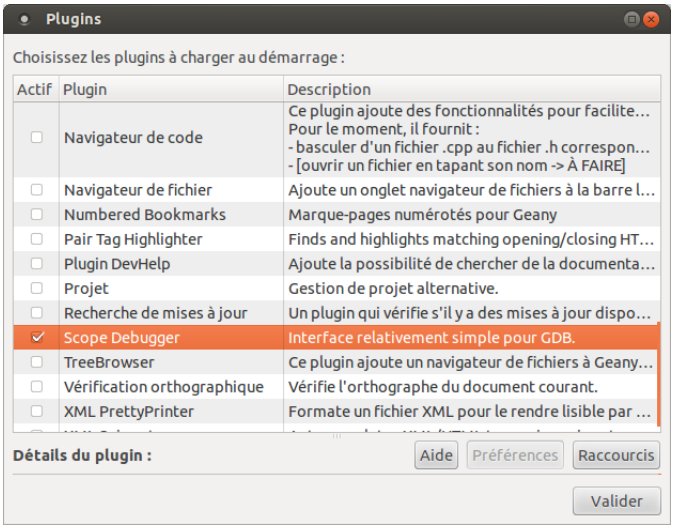
On constate que le calcul du prix TTC est défectueux !

geany-plugin-scope

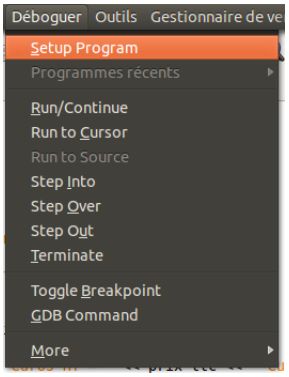
Aller dans le menu "Outils" puis dans "Gestionnaire de plugin " :



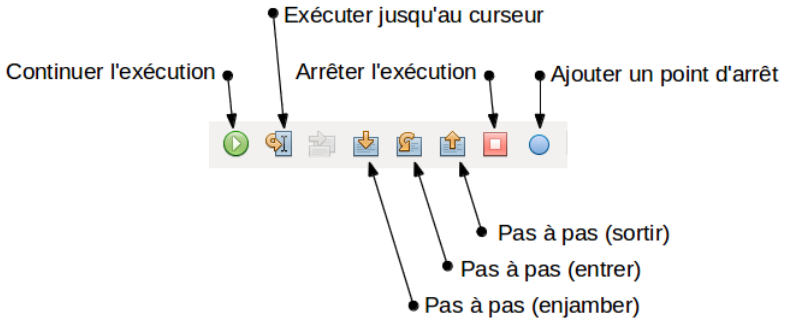
Activer le *plugin* Débogueur :



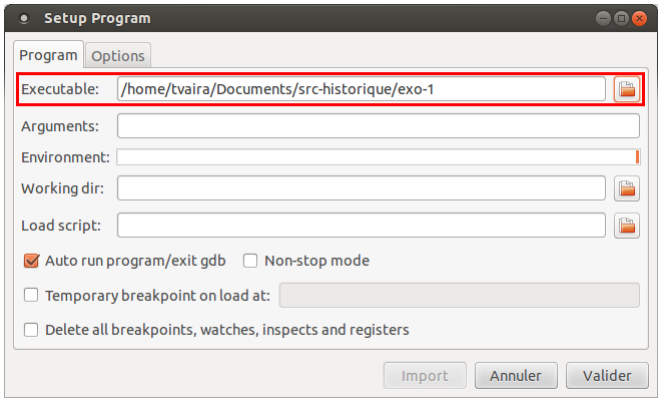
Vous disposez maintenant d'un menu dédié au débuguage :



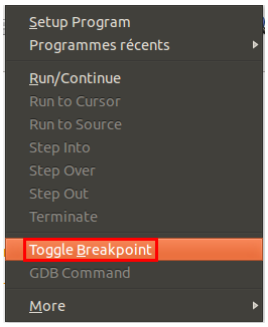
Et d'une barre d'outils :



Il faut charger l'exécutable à débuguer en sélectionnant "Setup program" :



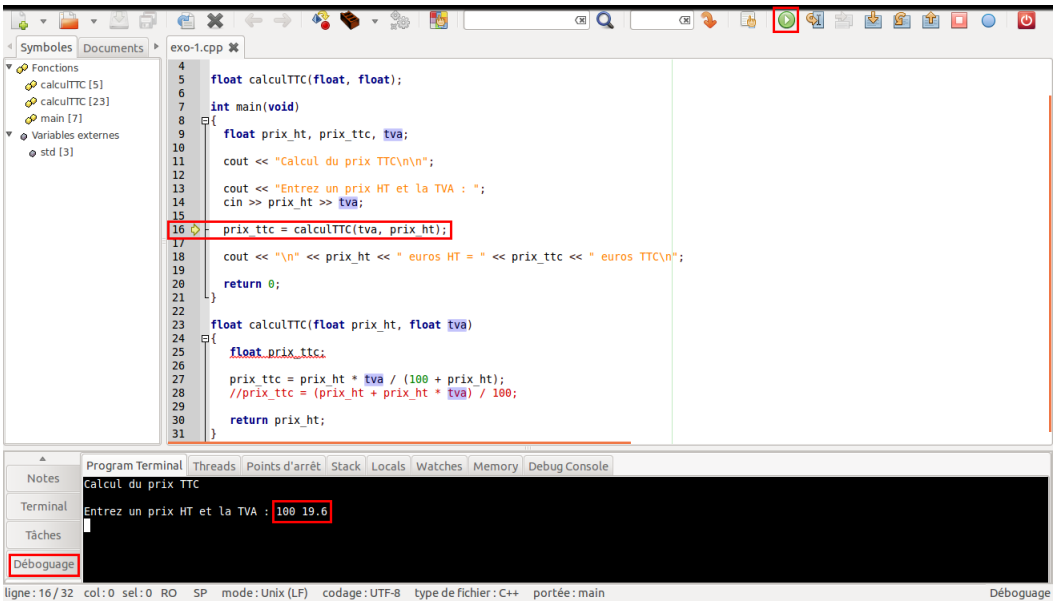
Vous pouvez placer des points d'arrêts en vous positionnant sur la ligne (ici la 16) puis en sélectionnant "Toggle Breakpoint" :



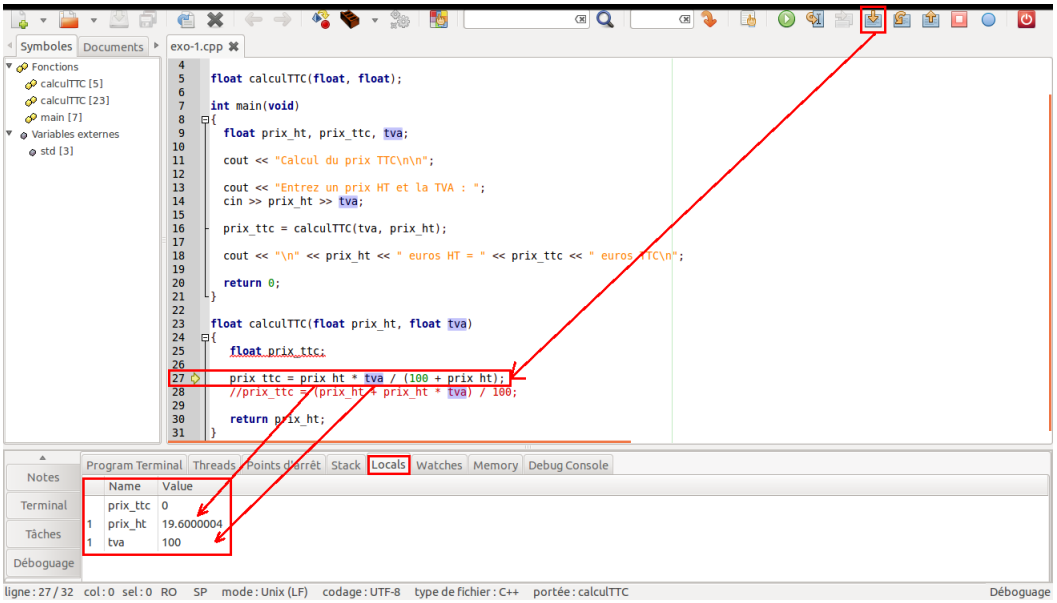

```
7   int main(void)
8   {
9     float prix_ht, prix_ttc, tva;
10
11    cout << "Calcul du prix TTC\n\n";
12
13    cout << "Entrez un prix HT et la TVA : ";
14    cin >> prix_ht >> tva;
15
16    prix_ttc = calculTTC(tva, prix_ht);
17
18    cout << "\n" << prix_ht << " euros HT = " << prix_ttc << " euros TTC\n";
19
20    return 0;
21  }
```

Puis démarrer l'exécution en sélectionnant "Run/Continue".

L'exécution se produit dans l'onglet "Program Terminal" et se poursuit jusqu'au point d'arrêt de la ligne 16 :

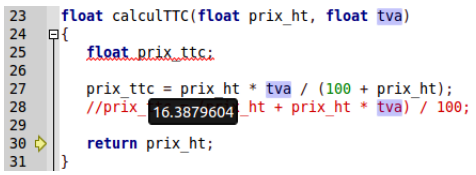


On sélectionne le mode d'exécution pas à pas (entrer *Step Into*) puis on observe les variables dans l'onglet "Locals" :



On constate que les variables `tva` et `prix_ht` ont été inversées au moment de l'appel à la fonction `calculTTC()` !

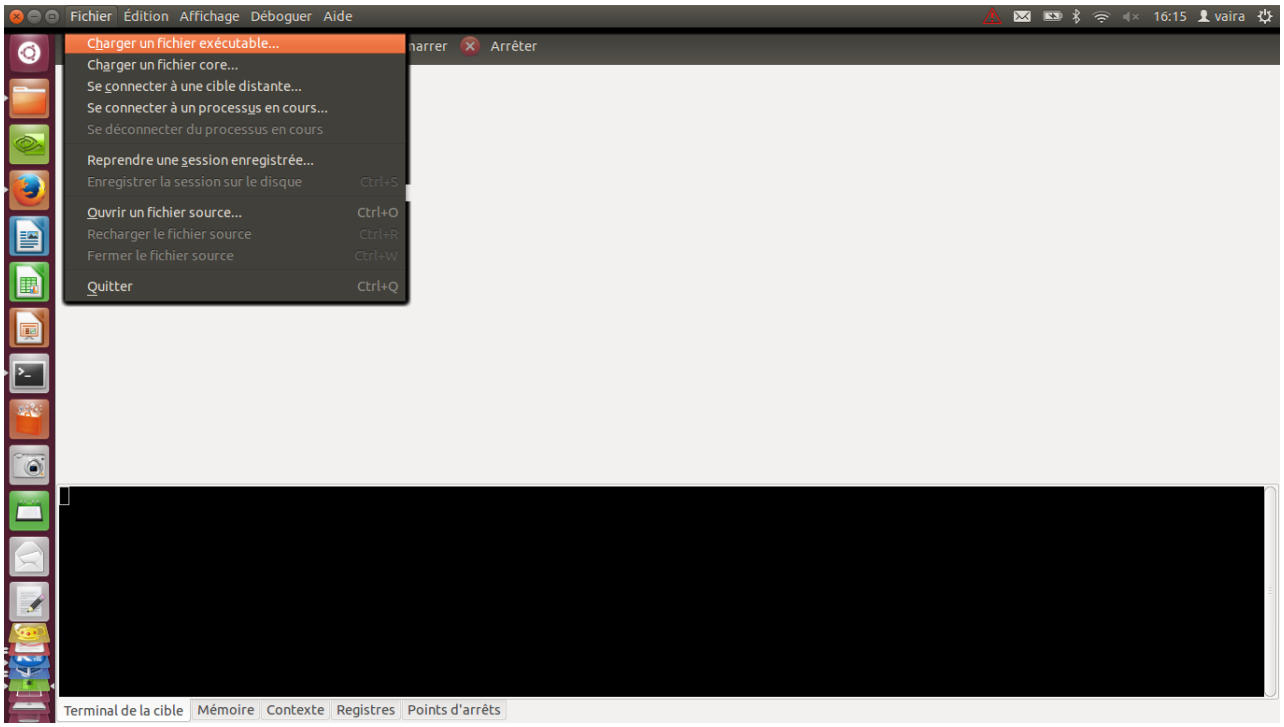
Il est aussi possible de placer le curseur de la souris sur une variable pour afficher sa valeur (ici la variable `prix_ttc`) :



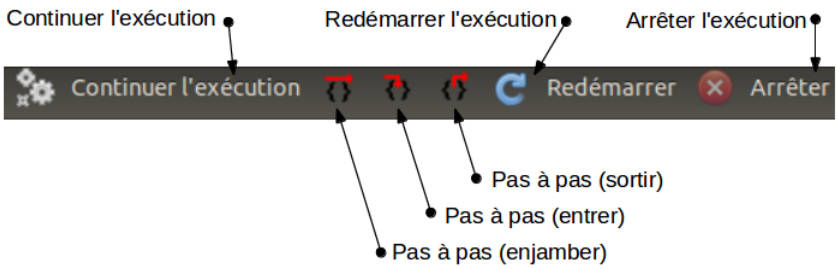
On constate que le calcul du prix TTC est défectueux !

nemiver

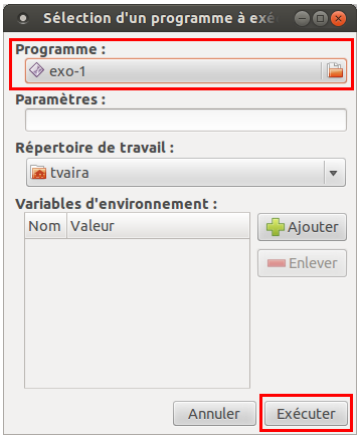
Vous disposez maintenant d'une application dédiée au débuguage :



Et d'une barre d'outils :

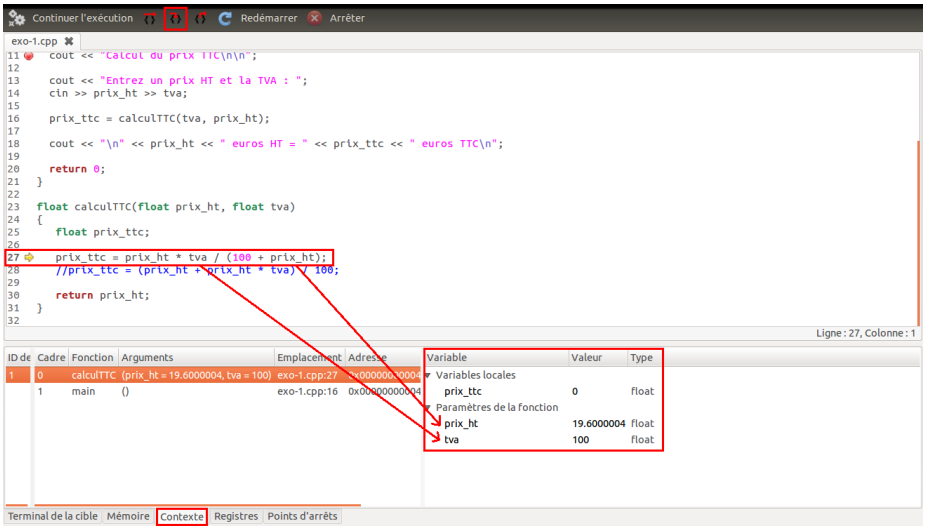


Il faut charger l'exécutable à débuguer en sélectionnant "Charger un fichier exécutable..." puis cliquer sur "Exécuter" :



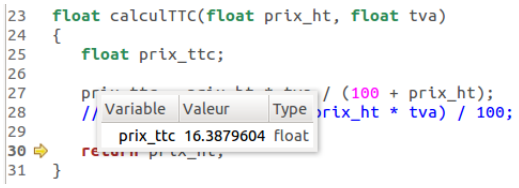
L'exécution se produit dans l'onglet "Terminal de la cible" et se poursuit jusqu'à la première instruction (ici ligne 11) :

On sélectionne le mode d'exécution pas à pas (entrer *Step Into*) puis on observe les variables dans l'onglet "Contexte" :



On constate que les variables `tva` et `prix_ht` ont été inversées au moment de l'appel à la fonction `calculTTC()` !

Il est aussi possible de placer le curseur de la souris sur une variable pour afficher sa valeur (ici la variable `prix_ttc`) :



On constate que le calcul du prix TTC est défectueux !

Question 1. En utilisant le débogueur de votre choix, détecter et corriger les 3 erreurs contenues dans le programme `exo-1.cpp`.

Exercice n°2 : gestion d'un historique d'évènements (ESI 2004)

Une ligne de fabrication est équipée de trois robots (R1, R2 et R3). Chaque robot est équipé de capteurs et d'actionneurs permettant le mouvement de la potence sur la ligne (horizontalement et verticalement).

Pour des raisons de maintenance du système (analyse de défaillances, vérification des temps de cycle, ...), il est nécessaire de conserver un **historique** d'activités regroupant les informations datées suivantes : **ordres** reçus, **réponses** transmises, **capteurs** détectés.

Le codage d'une information d'**ordre** comprend :
– un caractère parmi 'I', 'S', 'A', 'P', 'R' pour `Init()`, `Stop()`, `AllerA()`, `Pose()`, `Reprise()`
– un numéro de poste (mis par convention à 0 dans le cas des ordres 'I' et 'S')

Le **numéro de poste** sera implémenté comme suit :

```
typedef unsigned char TPoste;
```

Le codage d'une information **capteur** (position) comprend :
– un identifiant parmi 'Z', 'H', 'B', 'E', 'X', 'G', 'C', 'D'
– un numéro de poste (TPoste)

Le codage d'une information de **réponse** comprend seulement l'information suivante : POM (Prise Origine Machine) ou ACK (Acquittement) ou NAK (Non Acquittement).

Les réponses possibles émises par le robot sont implémentées comme suit :

```
typedef enum { POM = 0xFD, ACK, NAK } TReponse;
```

Question 2. Proposez une structure de données de type `TOrdre` assurant le codage d'une information d'**ordre**.

Question 3. De la même manière, donnez une définition du type `TPosition` assurant le codage d'une information **capteur**.

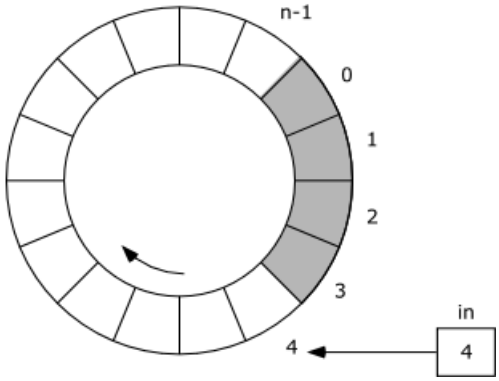
Question 4. Sachant que `TReponse` est un type `enum`, quelles seront les valeurs associées aux réponses ACK et NAK ?

Remarque : le codage de l'horodatage (une date et une heure) est réalisé par la fonction `time()` qui retourne le *timestamp* (de type `time_t`) qui représente le nombre de secondes écoulées depuis le début de l'époque UNIX (1er janvier 1970 00 :00 :00 GMT). Un *timestamp* est très pratique pour réaliser des traitements (calculs) sur des date/heure.

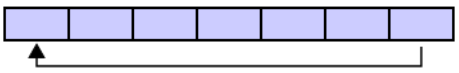
Question 5. Proposez la définition du type `TInfo` représentant un élément de l'historique. Cet élément doit regrouper une datation `date` suivie d'une **union** pouvant être une information d'**ordre**, de **capteur** ou de **réponse**.

La gestion de l'historique sera assurée par une classe `Historique` possédant les membres privés suivants :
– `unsigned int indice; // index du tableau historique`
– `const unsigned int taille; // le nombre maximum d'événements contenus dans le tableau historique`
– `TInfo *historique; // le tableau contenant les événements`

Le tableau `historique` est en réalité un **tampon circulaire** (*circular buffer*) assurant le stockage des événements les plus récents. Pour réaliser ce tampon (une zone mémoire), on utilise donc un tableau qui permettra de stocker des éléments de type `TInfo`.



L'utilisation circulaire d'un tampon permet de boucler la fin de la zone vers le début de celle-ci. Cela garantit aussi que l'on accède toujours à l'intérieur de la zone (on évitera ainsi les "erreur de segmentation").



Pour gérer ce tampon circulaire, on a besoin d'un index "`indice`" qui :
– permet le comptage des événements depuis l'initialisation,
– donne accès en permanence à l'emplacement suivant le dernier événement enregistré.

Voici l'ébauche de la classe `Historique` :

```
#define LG_HISTORIQUE 10

class Historique
{
  private:
    unsigned int indice;
    const unsigned int taille;
    TInfo *historique;

  public:
    Historique(int taille=LG_HISTORIQUE);
    ~Historique();

    unsigned int getNbEvenements() const;
    unsigned int getTaille() const;

    void ajouterOrdre(TOrdre ordre);
    //void ajouterPosition(TPosition position);
    //void ajouterReponse(TReponse reponse);

    void afficherDernierOrdre();
};
```

```
Historique::Historique(int taille/*=LG_HISTORIQUE*/) : indice(0), taille(taille)
{
  historique = new TInfo[taille];
}

Historique::~Historique()
{
  delete [] historique;
}

unsigned int Historique::getNbEvenements() const
{
   return indice;
}

unsigned int Historique::getTaille() const
{
   return taille;
}
```

C'est la méthode `ajouterOrdre(TOrdre ordre)` qui permettra d'enregistrer un **ordre** dans l'`historique`.

Question 6. Les extraits de code suivants proposent des solutions d'enregistrement dans le tableau "`historique`". Pour chaque extrait de code, préciser s'il répond aux besoins, justifiez en cas de réponse négative. **L'utilisation d'un débogueur est conseillée**.

```
// Solution 1 :
void Historique::ajouterOrdreSolution1(TOrdre ordre)
{
   historique[indice].date = time(NULL);
   historique[indice].infoOrdre = ordre;
   indice++;
}

// Solution 2 :
void Historique::ajouterOrdreSolution2(TOrdre ordre)
{
   if (indice >= taille ) indice = 0;
   historique[indice].date = time(NULL);
   historique[indice].infoOrdre = ordre;
   indice++;
}

// Solution 3 :
void Historique::ajouterOrdreSolution3(TOrdre ordre)
{
   historique[indice].date = time(NULL);
   historique[indice].infoOrdre = ordre;
   if (indice++ > taille ) indice = 0;
}

// Solution 4 :
void Historique::ajouterOrdreSolution4(TOrdre ordre)
{
```



```
   historique[indice % LG_HISTORIQUE].date = time(NULL);
   historique[indice % LG_HISTORIQUE].infoOrdre = ordre;
   indice++;
}
```

Les informations maintenues dans l'historique sont susceptibles d'être remontées vers un **système de supervision**. On doit pouvoir disposer de fonctions capables d'interroger son historique.

Question 7. Développer en quelques lignes la fonction `afficherDernierOrdre()` permettant l'affichage de la dernière information de type `TOrdre` datée dans l'historique. Pour afficher le `timestamp` de manière humainement lisible, vous pouvez utiliser la fonction `ctime()` ou `strftime()`.

Le programme d'essai :

```
#include <iostream>
#include <ctime>
#include <clocale>
#include <unistd.h>
#include "historique.h"

#define NB_ROBOTS    3
#define NB_EVENEMEMTS 16 /* pour les essais : changer cette valeur ! */

using namespace std;

int main() {
   Historique historique;
   char typesOrdre[] = { 'I','A','P','S','R' }; TOrdre ordreRobot; int i;
   time_t init; char horodatage[32];

   setlocale(LC_TIME, "fr_FR");
   init = time(NULL);
   strftime(horodatage, 32, "%d/%m/%Y à %T", std::localtime(&init));
   //cout << "Démarrage du système de journalisation le " << ctime(&init) << "\n";
   cout << "Démarrage du système de journalisation le " << horodatage << "\n";

   /* fabrique un historique */
   for(i = 0; i < NB_EVENEMEMTS; i++)
   {
      ordreRobot.poste = i%NB_ROBOTS + 1; /* simule un des 3 robots */
      ordreRobot.ordre = typesOrdre[i%5]; /* simule un ordre */
      historique.ajouterOrdre(ordreRobot);
      cout << "L'information TOrdre {" << ordreRobot.ordre << "," << (int)ordreRobot.poste
          << "} a été ajoutée à l'historique\n";
      sleep(1); /* simulons une attente avant le prochain événement à enregistrer */
   }

   //cout << "\n-> " << i << " informations TOrdre ont été consignées dans l'historique\n";
   cout << "\n-> " << historique.getNbEvenements() << " informations TOrdre ont été
       consignées dans l'historique\n\n";

   historique.afficherDernierOrdre();
   return 0;
}
```

test-historique.cpp

