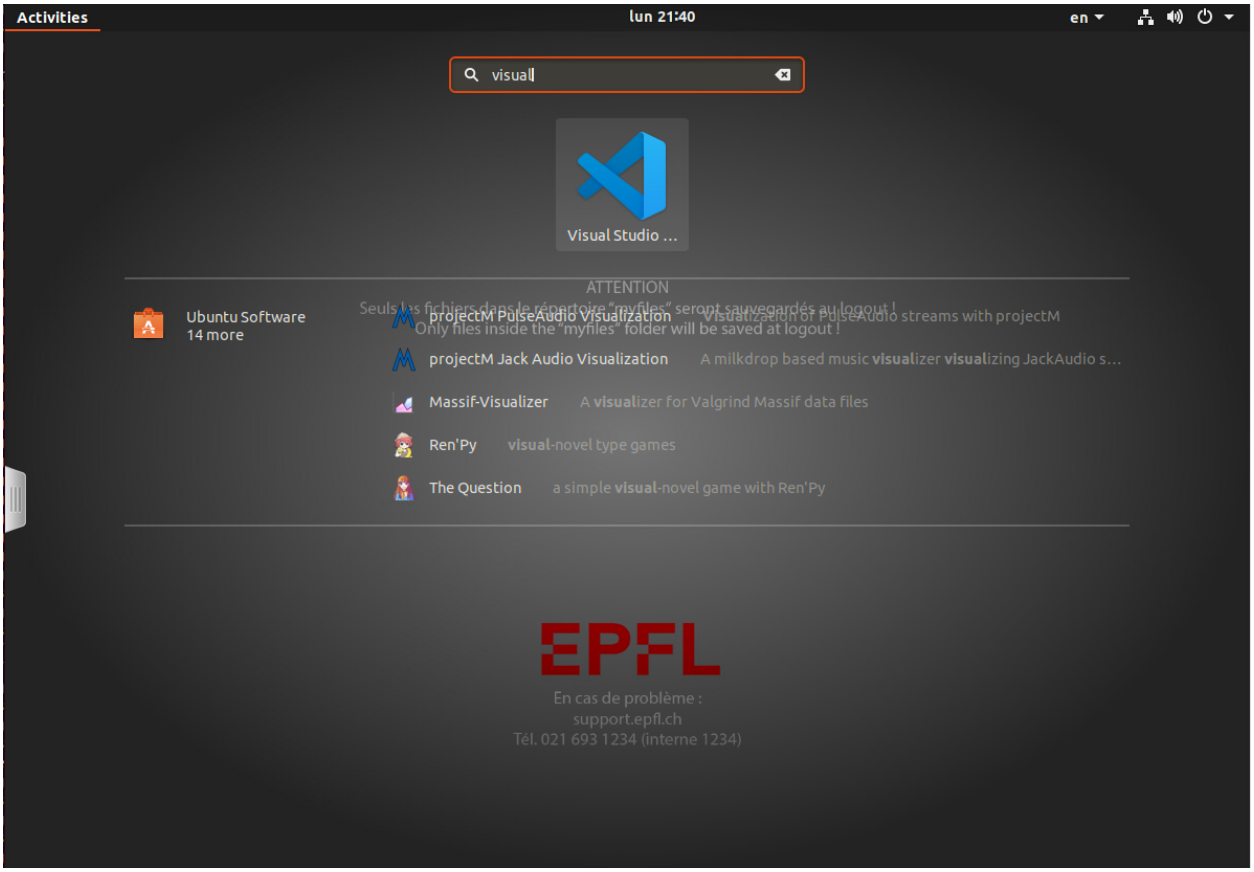
Session d'exercices – Mon premier programme

Cet exercice est le dernier exercice à faire pendant la première semaine sur la machine virtuelle. Vous devez avoir terminé l'exercice c_setup avant de commencer cet exercice. Les codes sources des programmes C sont, à l'instar de la plupart des langages de programmation, de simples fichiers texte. N'importe quel éditeur de texte permet donc d'écrire un programme en C. Nous vous proposons d'utiliser *Visual Studio Code*. L'un des avantages de cet éditeur est qu'il possède un mode C agréable (il présente les programmes avec une mise en page et une coloration adaptées ; on appelle cela la *mise en évidence syntaxique*).

Le but de cet exercice est justement de vous apprendre, pas à pas, à

- paramétrer l'éditeur ;
- compiler un programme C ;
- corriger les erreurs de compilation.

Pour lancer cet éditeur, il faut chercher «Visual Studio Code» parmi toutes les applications disponibles.



Étape 1 : Paramétrer l'éditeur et les extensions

Commencez par choisir un thème de couleur pour votre éditeur via *File→Preferences→Color Theme*. Ici, nous utilisons **"Dark + (default dark)"**.

Ensuite, cliquez sur l'icône correspondant aux extensions sur le côté gauche. Cherchez dans le champ les extensions suivantes et installez-les :

- **C/C++** par Microsoft. (Cet extension est probablement déjà installée.)

  Plus d'info: *C/C++ support for Visual Studio Code is provided by a Microsoft C/C++ extension to enable cross-platform C and C++ development using VS Code on Windows, Linux, and macOS. The focus is on code editing, navigation, and debugging support for C and C++ code everywhere that VS Code runs.* lien

- **Code Spell Checker** par Street Side software

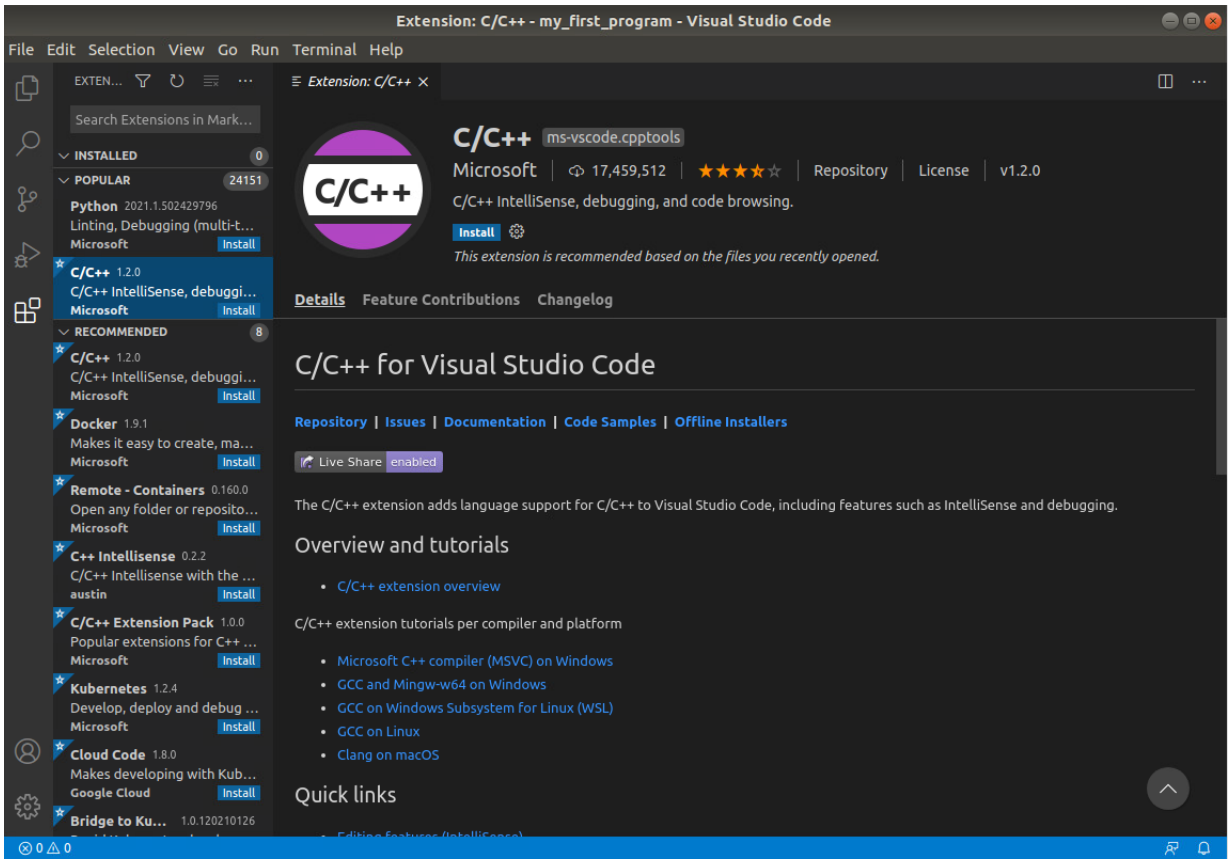
- **indent-rainbow** par oderwat

  Plus d'info: *This extension colorises the indentation in front of your text alternating four different colors on each step.* lien

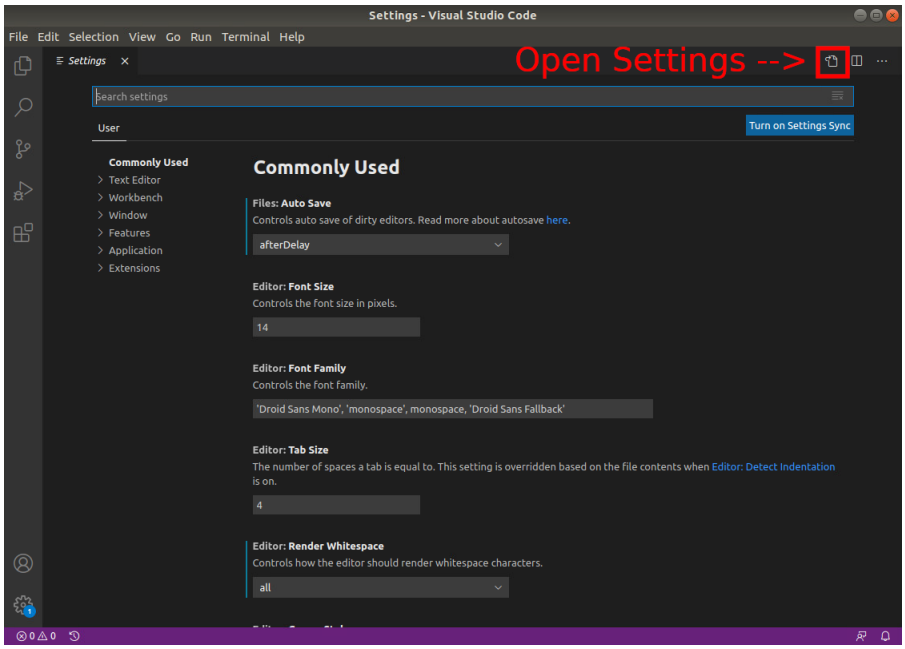
- **Rainbow Brackets** par 2gua

  Plus d'info: *Provide rainbow colors for the round brackets, the square brackets and the squiggly brackets. The isolated right bracket will be highlighted in red.*

Exemple pour C/C++ de Microsoft :



A présent, configurez de l'éditeur de la façon suivante : parcourez *File→Preferences→Settings*. Une fenêtre comme celui-ci s'ouvrira :



Cliquez comme indiqué ci-dessus afin d'ouvrir le fichier de configuration. Ensuite, copiez le texte suivant

```
"editor.renderWhitespace": "all",
"editor.insertSpaces": true,
"files.autoSave": "afterDelay",
"files.autoSaveDelay": 60000,
"indentRainbow.includedLanguages": [],
```

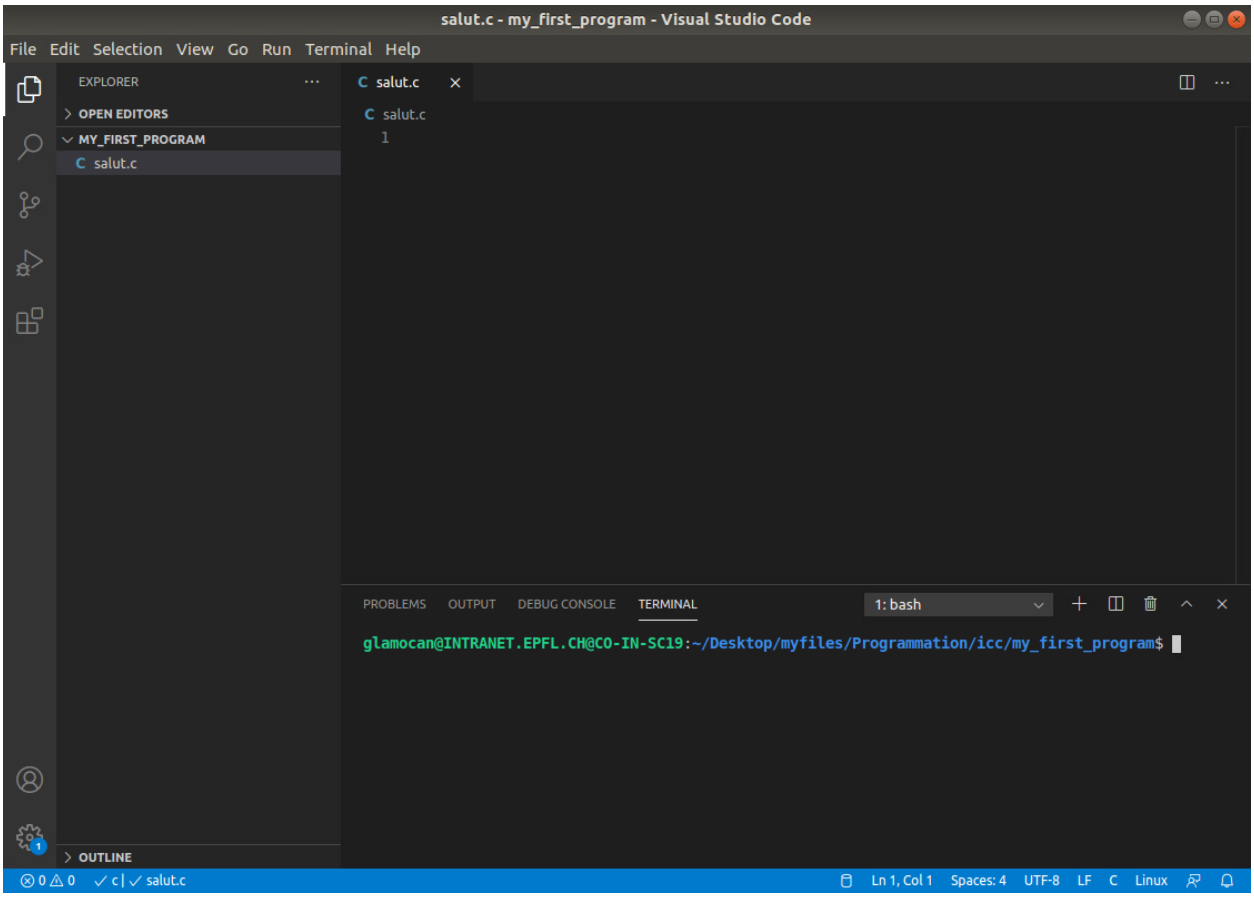
et collez-le entre l'accolade ouverte et l'accolade fermée du fichier settings.json.

(Si un réglage semble incorrect, vérifiez la syntaxe ou s'il est lié à une extension que vous venez d'installer, essayez de la réactiver en cliquant sur ***Reload****).*

Étape 2 : Créer et modifier un programme

Rappelez vous les commandes Unix `mkdir` et `touch`. Utilisez-les pour naviguer vers le répertoire `Programmation/icc` et créez-y un répertoire appelé `my_first_program`. Dans ce répertoire, créez le fichier `salut.c`.

Ensuite, depuis Visual Studio Code, ouvrez ce dossier (*File→Open Folder*). À coté gauche de votre éditeur Visual Studio Code, vous trouverez tous les fichiers du répertoire `my_first_program`, `salut.c` étant le seul pour le moment. En plus, dans le fenêtre `TERMINAL` en bas à droite, le chemin vers le répertoire est déjà prêt. Ouvrez le fichier `salut.c` dans l'éditeur Visual Studio Code.

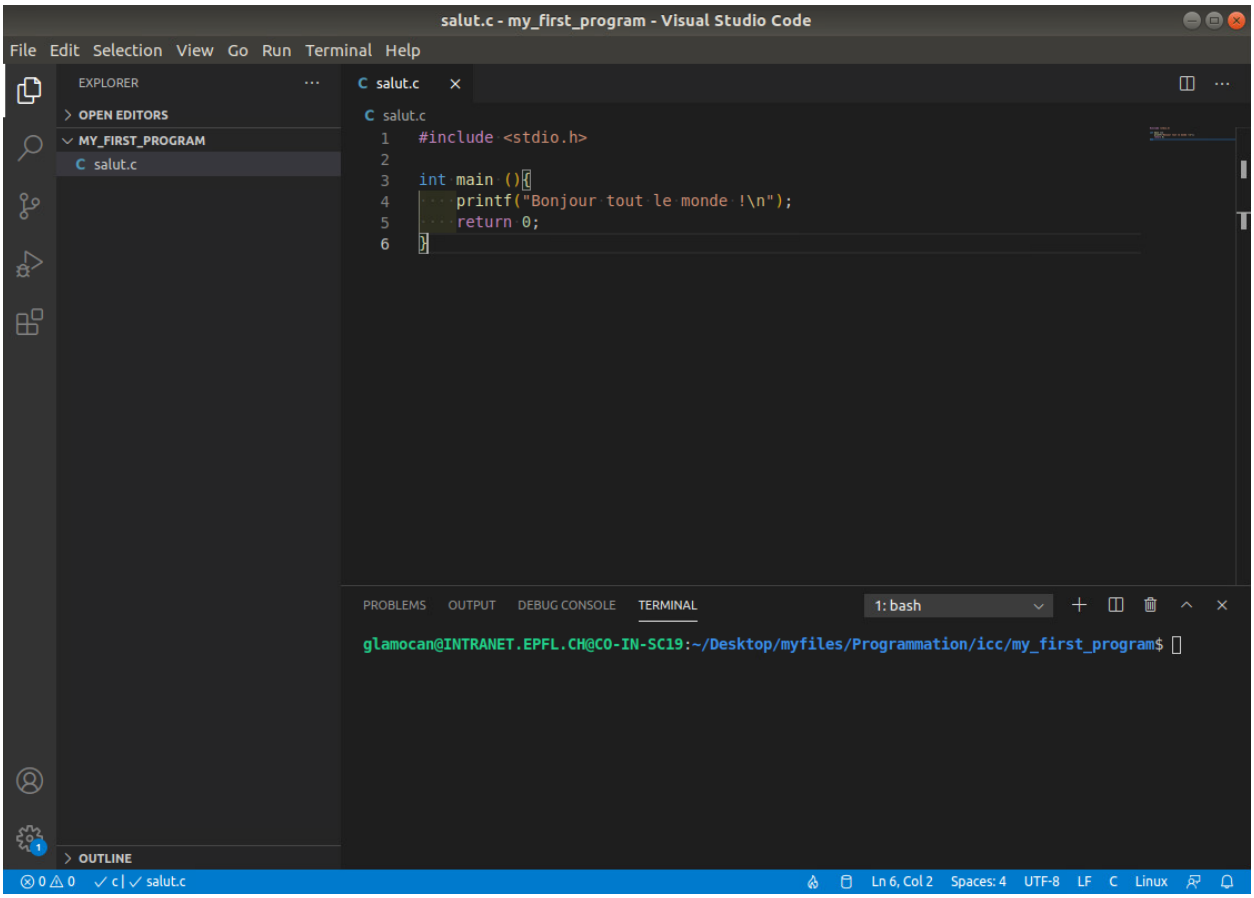


Dans votre fichier `salut.c`, ajoutez le petit programme suivant (ne faites pas copier-coller, entrez le code par vous même):

```c
#include <stdio.h>

int main() {
  printf("Bonjour tout le monde !\n");
  return 0;
}
```

Cela devrait ressembler à quelque chose comme cela :



Remarque : Certains mots apparaissent colorés. Il s'agit de mots ayant des fonctions syntaxiques particulières, et leur couleur peut améliorer la lisibilité des programmes. Sauvegardez le programme : **Ctrl-s** ou via le menu *File→Save*.

Structure du programme :

- La ligne 1 est nécessaire pour indiquer au programme que vous allez utiliser les fonctions de la bibliothèque standard (`stdio`), par exemple `printf` ou `scanf`.
- La ligne 3 indique le début du programme. Les instructions comprises entre l'accolade ouvrante { et l'accolade fermante associée } constituent la définition du programme principal (la fonction `main`), c'est-à-dire les instructions qui seront effectivement exécutées au lancement du programme.
- La ligne 4 affiche Bonjour tout le monde ! à l'écran (au terminal).
- La dernière instruction du programme est l'instruction `return 0`, qui renvoie zéro comme *le code de sortie de notre programme*. **L'utilisation du code de sortie 0 est une convention pour signaler que l'exécution du programme a réussi.**

Avant d'être exécuté, un programme C doit être **compilé** : le fichier texte contenant *le code source* doit être traduit en *langage machine*. On effectue cette opération à l'aide d'une commande de compilation, qui peut être tapée dans une fenêtre Terminal ou, selon l'éditeur, grâce à un bouton dédié. Ici, nous utiliserons le terminal de Visual Studio Code.

L'**autocomplétion** est un outil très pratique : grâce à la touche Tab, le terminal propose de compléter ce que vous écrivez par un **nom de fichier existant** dans le répertoire courant. Appuyez une fois sur Tab pour compléter un mot et deux fois sur Tab pour consulter les possibilités d'autocomplétion.

Dans le terminal, tapez la commande suivante en utilisant l'autocomplétion pour *salut.c* (puis appuyez sur *Enter*) :

```
gcc -Wall salut.c -o salut
```

Si rien ne s'écrit, c'est que tout s'est bien passé. Observez dans votre dossier (ou bien utilisez la commande ls sal* pour voir tout les fichiers dont le nom commence par **sal** : il existe désormais un fichier nommé *salut* (sans .c) : c'est l'**exécutable**. Si vous travailler sur Windows, le nouveau fichier est *salut.exe*.

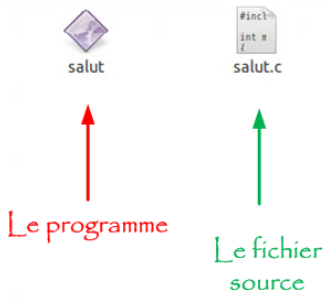
Il a été créé grâce à la commande que nous venons de taper :

`gcc` est un logiciel libre utilisé pour compiler divers langages de programmation.

`-Wall` est une option de gcc pour demander l'affichage des avertissement (warnings).

`-o` est une option de gcc pour sauvegarder le résultat (notre programme exécutable) dans le fichier dont le nom est écrit juste après. Donc, le nouveau programme sera appelé `salut` et sera placé dans le même répertoire que le fichier source `salut.c`!

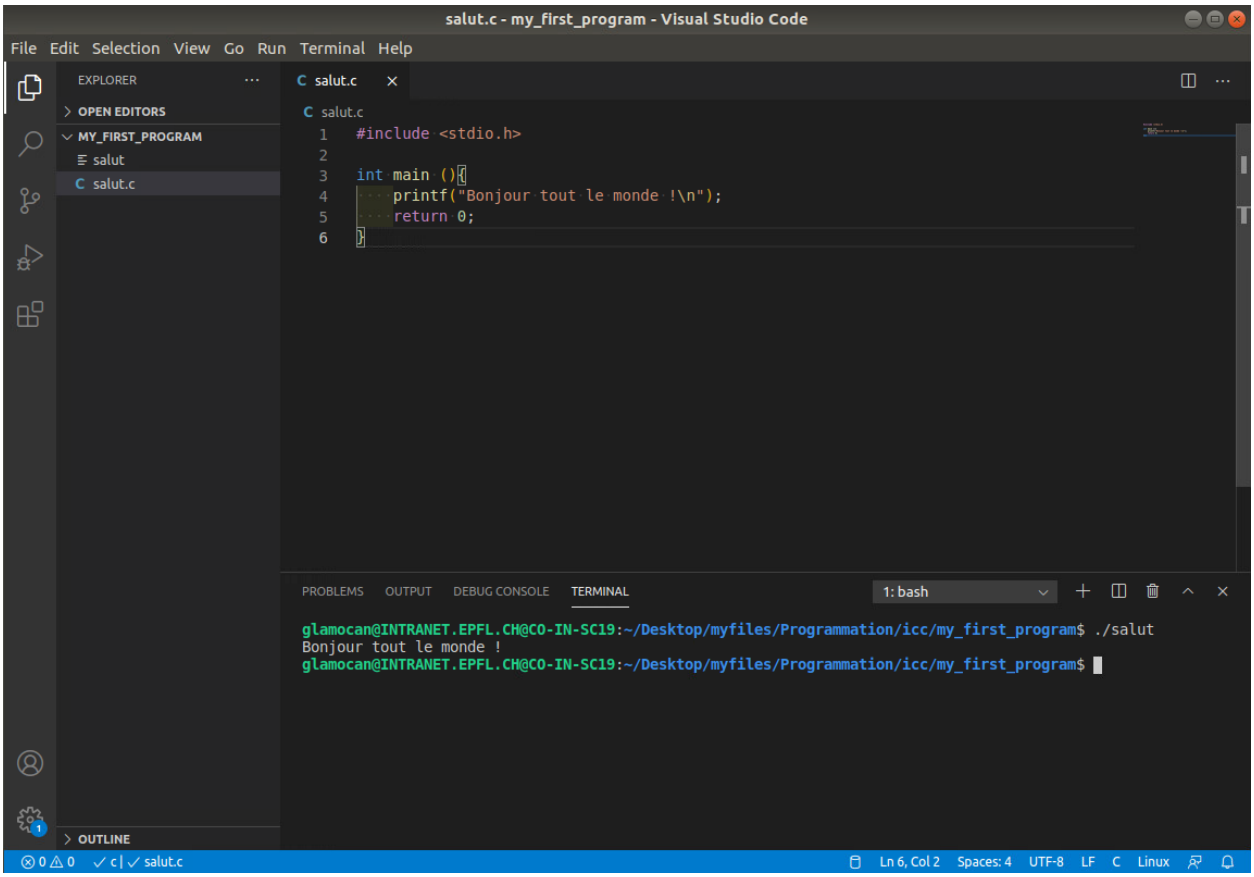
`salut.c` est le nom du fichier source.



Si la compilation s'est déroulée sans problème, il faut tester votre programme. Toujours dans le terminal, dans le répertoire `my_first_program` (parce que votre programme se trouve là), lancez le en tapant

```
./<nom_du_programme>
```

Attention: sans les crochets et sans l'extension .c ! Alors, dans cet exemple-ci, `<nom_du_programme>` doit être remplacé par `salut`. Ensuite, le message Bonjour tout le monde ! devrait apparaître dans le terminal.



On utilise encore l'autocomplétion... En faisant attention à ce qu'elle nous propose !

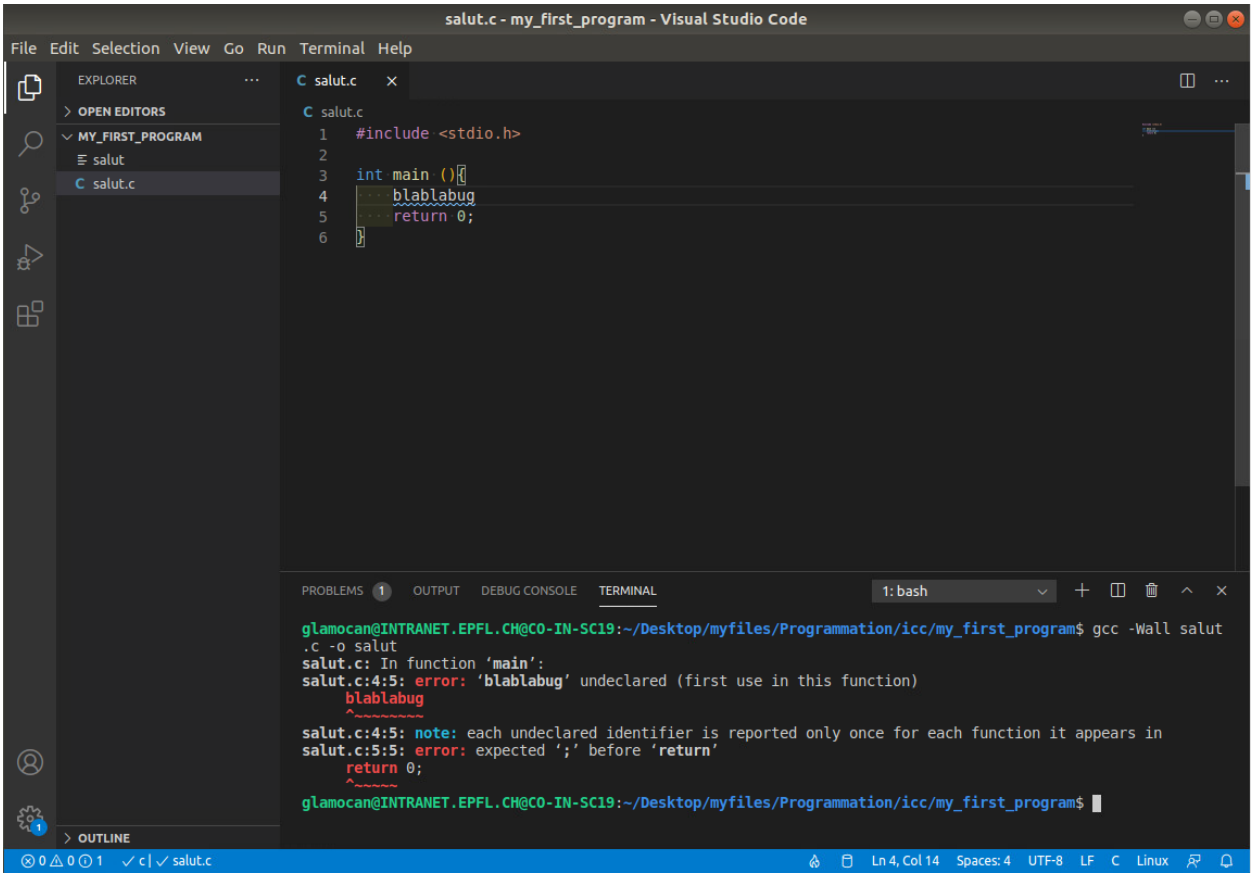
Étape 3 : Erreurs de compilation

Nous allons maintenant nous intéresser à la correction des erreurs d'un programme, en tout cas celles que le compilateur peut détecter. Cela vous arrivera souvent. Heureusement, le compilateur `gcc` est assez fort pour trouver les erreurs. Apprenez donc à les comprendre. Il existe divers types d'erreurs :

1. **Les erreurs de syntaxe** : le programme est mal écrit et le compilateur ne le comprend pas. Ces erreurs sont relativement faciles à trouver, car le compilateur signale toujours le problème, en indiquant souvent l'endroit de l'erreur.
2. **Les erreurs d'implémentation** : la syntaxe du programme est correcte (il compile), mais ce qu'il fait est erroné (par exemple une division par zéro se produit, ou une variable n'a pas été initialisée correctement). Ce type d'erreur ne se détecte que lors des tests du programme (soit par un arrêt intempestif (cas de la division par zéro), soit par des résultats erronés (cas de la mauvaise initialisation)).
3. **Les erreurs d'algorithme** : l'algorithme implémenté ne fait pas ce que l'on croit (ce qu'il devrait). Ce type d'erreur est en fait assez proche du précédent. Cependant, c'est plus la méthode globale qui est erronnée, alors que dans le cas précédent, l'erreur provient le plus souvent d'une étourderie ou d'un manque de précision dans une des étapes du codage de l'algorithme.

4. **Les erreurs de conceptions** : ici c'est carrément l'approche du problème qui est erronnée, souvent en raison d'hypothèses trop fortes ou d'hypothèses non explicitées.

Introduisez volontairement une erreur de syntaxe dans votre programme (supprimez par exemple la dernière accolade ou le dernier point-virgule). Relancez ensuite la compilation. Dans le terminal, vous devriez obtenir un message indiquant le fichier où se trouve l'erreur (salut.c), la ligne concernée, ainsi que le type d'erreur :



Ici, on nous signale deux erreurs. La première est que le compilateur ne sait pas ce que veut dire "blablabug", il ne connaît aucune entité préalablement déclarée sous ce nom. La seconde est qu'il manque un point-virgule.

Attention: assez souvent, la source d'erreur **ne se trouve pas exactement sur la ligne signalée**, mais sur la ligne précédente ou la ligne suivante. Par exemple ici, on nous signale une erreur à la ligne 5 mais en lisant le message d'erreur, on voit qu'il s'agit du '**;**' manquant à la ligne 4.

Il faut toujours lire tous les messages de compilation et d'exécution, qu'il s'agisse de *warnings* (non fatals, à la compilation) ou d'*erreurs*. Ils sont généralement explicites sur ce que le compilateur a essayé de faire/a compris de ce que vous avez écrit. Il est conseillé de toujours traiter les erreurs dans l'ordre car souvent, de la première découlent les suivantes. A l'inverse, une erreur peut aussi en cacher d'autres qui n'apparaîtront qu'une fois la première résolue !

Corrigez les erreurs, et relancez la compilation.