パート 2: Ubuntu* で C++ 開発向けに Visual Studio* Code を設定

この記事は、Codeplay Blogs に公開されている「Setting up C++ development with Visual Studio® Code on Ubuntu」の日本語参考訳です。原文は更新される可能性があります、原文と翻訳文の内容が異なる場合原文を優先してください。

2023 年 3 月 1 日

本シリーズの記事:

- パート 1: DPC++ と Visual Studio* Code による SYCL* のデバッグ
- パート 3: Ubuntu* で SYCL* 開発向けに oneAPI、DPC++、Visual Studio* Code を設定

パート 2 では、Ubuntu* 上の Visual Studio* Code で C++ コードを記述し、コンパイルしてデバッグする方法を説明します。パート 3 では、SYCL* 開発向けに Visual Studio* Code を設定する方法を説明します。本ガイドでは、Visual Studio* Code の IDE を使い慣れていないことを前提に解説します。

まず、Visual Studio* Code をインストールします。インストール方法については、こちらのガイド (英語) を参照することを推奨します。

新規インストールした Visual Studio* Code は、実質的に空の IDE シェルであるため、開発ニーズに合わせて設定する必要があります。表 1 に C/C++ 開発で使用できる Visual Studio* Code の拡張機能を示します。Visual Studio* Code の [拡張機能] パネルで、次の拡張機能を検索してインストールします。「推奨」と記載されている拡張機能はオプションであり、別のビルド構成からプロジェクトを移行する場合や、複数のビルド構成を使用する場合に、プロジェクトのナビゲーションを支援します。

Visual Studio* Code の拡張機能	Microsoft* C/C++ 要件	注
Microsoft* C/C++ Extension Pack	○	
CMake*	推奨	一部のインテルの DPC++ サンプルコードは、CMake* プロジェクト構成を使用します。この記事では、CMake* は使用しません。
Makefile Tools	推奨	一部のインテルの DPC++ サンプルコードは Makefile プロジェクト構成を使用します。この記事では、Makefile は使用しません。
C/C++ Runner	推奨	Debug ビルドと Release ビルドのビルド構成を自動化します。この記事では使用しません。
.md ドキュメントリーダー	推奨	ほとんどのサンプルコードには、.md 形式の Readme やドキュメントがあります。Visual Studio* Code で **Ctrl + Shift + v** キーを押して、.md ドキュメントを表示します。

表 1: Visual Studio* Code の C/C++ ヘルパー拡張

このほかにも、以下のような推奨パッケージがありますが、本記事では使用しません。

- Git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git (英語)
- CMake: ターミナルウィンドウで `sudo apt update` を実行し、`sudo apt -y install cmake pkg-config build-essential` を実行します。

C/C++ コンパイル向けに Visual Studio* Code を設定

すでに標準の C/C++ 開発環境が設定されており、Visual Studio* Code で作業している場合は、Helloworld のビルドとデバッグの説明に進みます。

> **注:**
> Visual Studio* Code の C/C++ プロジェクトやワークスペースは、**フォルダー**と呼ばれます。フォルダーには、プロジェクトのファイルとサブフォルダーが含まれます。
>
> Microsoft* の Visual Studio* Code 案内ビデオの多くは、Ubuntu* の新規インストールでは利用できないコーデックを必要とします。必要なコーデックを入手するには、**ubuntu-restricted-extras** インストール・パッケージを検索してください。
>
> Ubuntu* を新規にインストールすると、**gcc** コンパイラー (**g++** や **g++9** と呼ばれることもある) がすでにインストールされており、システム上のいくつかのフォルダー (/usr/bin/... や /bin/...) に存在します。Visual Code* Studio の C/C++ 拡張で利用可能なコンパイラーをリストすると、いくつかの gcc コンパイラーが表示されますが、これらはすべて 1 つの gcc 実行ファイルのエイリアスです。gcc のデバッガーは **gdb** と呼ばれます。
>
> デフォルトでは、Microsoft* C/C++ 拡張は OS 環境変数 **PATH** から gcc コンパイラーを自動検出します。インテル® oneAPI DPC++ コンパイラーなど、その他のコンパイラーは PATH に含まれている可能性がありますが、新しいビルド構成を作成する際にはリストされません。リストにインテル® コンパイラーを追加する方法は、パート 3 の「**Visual Studio* Code の C/C++ プロジェクトを DPC++ プロジェクトに変換する準備**」を参照してください。

C/C++ 開発に必要な Visual Studio* Code の拡張機能をインストールしたら、C/C++ プロジェクト向けに Microsoft* C/C++ 拡張を設定する必要があります。Microsoft* C/C++ 拡張は、プロジェクト・フォルダーに **.vscode** サブフォルダーを作成して、必要なファイルを配置します。これらはプロジェクトの構成ファイルであり、プログラムのコンパイルとデバッグに関する情報が含まれています。

表 2 は、Microsoft* C/C++ 拡張によって生成される **.json** 構成ファイルです。

構成ファイル	場所	説明	依存関係
tasks.json	.vscode ワークスペース・フォルダー	1 つ以上のビルド構成を含み、各構成には実行形式ごとのビルド設定が含まれます。 構成ごとに特定のコンパイラーを指定できます。	必須
launch.json	.vscode ワークスペース・フォルダー	1 つ以上のデバッグ構成を含み、各構成には実行形式ごとのデバッグ設定が含まれます。	必須
c_cpp_properties.json	.vscode ワークスペース・フォルダー	コンパイラーのパスと C/C++ IntelliSense 設定を含みます。	必須
settings.json	.vscode ワークスペース・フォルダー	構成ファイルの変数を含みます。 変数は、Visual Studio* Code の一部のオプション (すべてではない) でテキスト置換を可能にするテキスト値を保持します。 オプションの $env 変数が動作しない場合の代替です。	プロジェクトの要件と構造に依存します。

表 2: Microsoft* C/C++ 拡張のプロジェクトの構成ファイル

図 1a および 1b に示す **Helloworld** プログラムを使用して、デフォルトの **gcc** コンパイラーとデバッガーを使用した通常の C++ プロジェクトを設定する手順を説明します。表 2 の構成ファイルを編集して、デバッグ実行ファイルとリリース実行ファイルをビルドし、Visual Studio* Code のコマンドパレットから実行ファイルを選択できるようにします。任意のエディターを使用して、Helloworld プログラムのプログラムファイルを作成します。

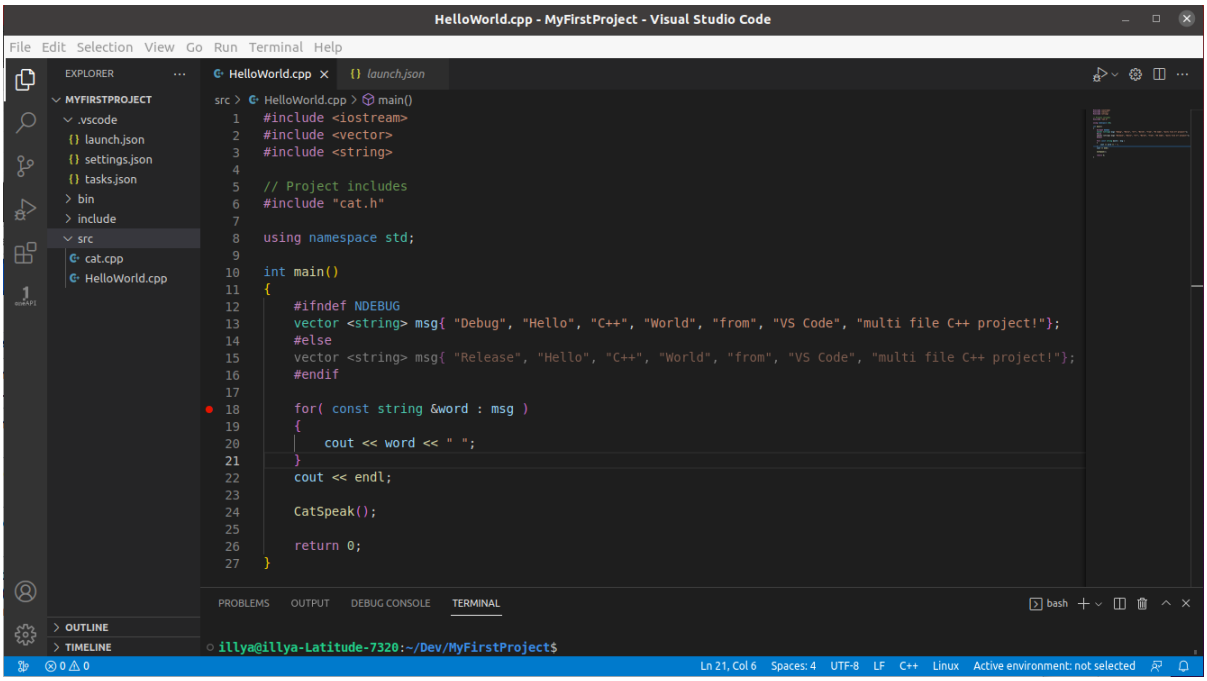

図 1a: Helloworld.cpp

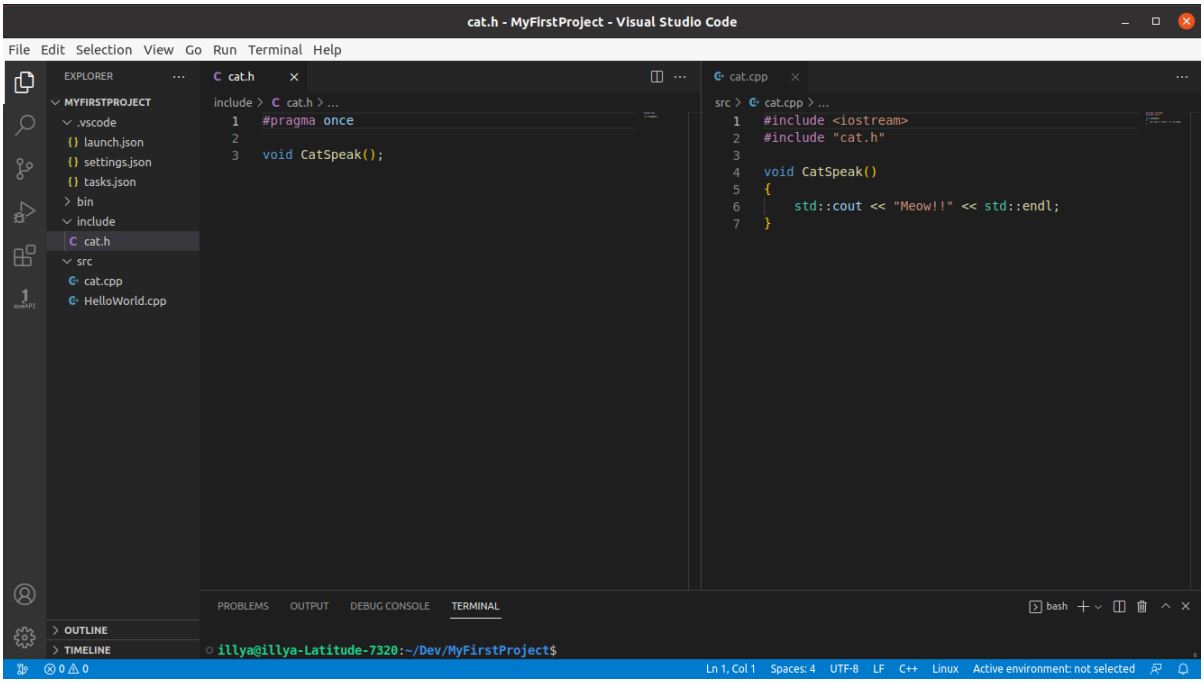

図 1b: cat.h と cat.cpp

プロジェクトの構造

図 2 は、ワークスペース・フォルダーの典型的なファイル構造を示します。このファイル構造を任意の場所に作成し、Helloworld コードファイルを適切なフォルダーに配置します。この例では、プロジェクトの名前を **MyFirstProject** (フォルダー) とします。MyFirstProject フォルダーに Helloworld という名前の C/C++ プロジェクトを作成します。

ターミナルウィンドウを開いて、プロジェクト・フォルダーである MyFirstProject 以下に移動します。ターミナルのコマンドラインに `code .` と入力すると、まだ開いていない場合は Visual Studio* Code が開き、すでに開いている場合は新しいアプリケーション・ウィンドウが開きます。

プロジェクトを再度開く際に、Visual Studio* Code は、ワークスペース・フォルダーを選択するように求めます。図 1a では、[エクスプローラー] ペインに隠しフォルダーである .vscode フォルダーが表示されていますが、それ以外のフォルダー構造は 図 2 と同じです。C/C++ 拡張では、.vscode フォルダーが自動的に作成されます。プロジェクトに関連する動作属性のほとんどは、このフォルダーにあるファイルに保持されます。

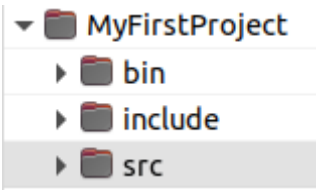

図 2: プロジェクト・フォルダーの構造

プロジェクトのビルドとデバッグ構成の設定では、ファイルや出力ファイルを見つけられるように、このファイル構造に対応した正しいデスティネーション・フォルダーを設定する必要があります。

プロジェクトの複数のビルド構成の設定

C++ プロジェクトの設定では、最初に必要なビルドの種類を設定します。このプロジェクトでは、**bin** フォルダーにデバッグ実行ファイルとリリース実行ファイルを配置します。2 つの実行ファイルを区別するため、デバッグ実行ファイルの名前にはサフィックス **_d** を追加します。任意のファイル名を指定できます。C/C++ 拡張によって生成される task.json ファイルをカスタマイズして、これらの設定を行います。

Visual Studio* Code のメニューから **[ターミナル] > [規定のビルド タスクの構成…]** を選択します。ドロップダウン・リストから gcc コンパイラーを選択します。前述のとおり、リストされている gcc コンパイラーはすべて 1 つのコンパイラーのエイリアスです。タスクを含む新しい **tasks.json** ファイルが生成されます。タスクには、ドロップダウン・リストから選択したコンパイラーの名前が付けられており、[エクスプローラー] ペインで選択した **.cpp** ファイルを、デバッグメタデータを含む実行ファイル (デバッグ実行ファイル) にビルドする gcc 引数が含まれています。

> **注:**
> Visual Studio* Code のメニューから **[ターミナル] > [タスクの構成...]** を選択し、ドロップダウン・リストからコンパイラーを選択して、同じ操作を行うことができます。
>
> Visual Studio* Code のオンライン・ドキュメントでは、C/C++ プロジェクトの .json ファイルで使用可能な C/C++ オプションがすべて説明されています。
> https://code.visualstudio.com/docs/editor/variables-reference (英語)
>
> tasks.json ファイルがすでに存在する場合、**[規定のビルド タスクの構成...]** を選択すると、Visual Studio* Code は新しい設定を既存のタスクの最後に追加するので、既存のタスクが破損することはありません。

しかし、プロジェクトに複数のコードファイルが含まれており、デバッグとリリースのビルドを別々に行うには、要件に合うように tasks.json を編集する必要があります。図 4a と図 4b は、このプロジェクトに使用するビルド構成タスクです。参考までに、図 3 は Microsoft* C/C++ 拡張によって作成されるデフォルトのビルド構成です。

```
{
    "type": "cppbuild",
    "label": "C/C++: g++ build active file",
    "command": "/usr/bin/g++",
    "args": [
        "-fdiagnostics-color=always",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}"
    ],
    "options": {
        "cwd": "${fileDirname}"
    },
    "problemMatcher": [
        "$gcc"
    ],
    "group": {
        "kind": "build",
        "isDefault": true
    },
    "detail": "compiler: /usr/bin/g++"
}
```

図 3: Microsoft* C/C++ 拡張によって作成されるデフォルトのビルド構成

```
{
    "type": "cppbuild",
    "label": "Debug C/C++: g++ build active file",
    "command": "/usr/bin/g++",
    "args": [
        "-fdiagnostics-color=always",
        "-g",
        "-I${workspaceFolder}/include",
        "${workspaceFolder}/src/*.cpp",
        "-o",
        "${workspaceFolder}/bin/${config:programName}_d"
    ],
    "options": {
        "cwd": "${fileDirname}"
    },
    "problemMatcher": [
        "$gcc"
    ],
    "group": "build",
    "detail": "compiler: /usr/bin/g++"
},
```

図 4a: Helloworld デバッグ実行ファイルをビルドするカスタム構成タスク

```
{
    "type": "cppbuild",
    "label": "Release C/C++: g++ build active file",
    "command": "/usr/bin/g++",
    "args": [
        "-DNDEBUG",
        "-I${workspaceFolder}/include",
        "${workspaceFolder}/src/*.cpp",
        "-o",
        "${workspaceFolder}/bin/${config:programName}"
    ],
    "options": {
        "cwd": "${fileDirname}"
    },
    "problemMatcher": [
        "$gcc"
    ],
    "group": "build",
    "detail": "compiler: /usr/bin/g++"
},
```

図 4b: Helloworld リリース実行ファイルをビルドするカスタム構成タスク

注:
tasks.json ファイルには、複数のタスクを任意の順序で含めることができます。

プロジェクト要件に合わせてビルド構成をカスタマイズするには、以下の操作を行います。

1. デフォルトのデバッグ構成を編集して図 4a のようにします。
2. デバッグ・ビルド・タスクを複製して、図 4a の最初のデバッグタスクの下に挿入します。これがリリース・ビルド・タスクになります。
3. 複製を編集して、図 4b に示すリリース・ビルド・タスクにします。

デフォルトのデバッグ構成からの注目すべき変更点は、以下のとおりです。

> **注:**
> 次のタスクを追加するため、最後の中括弧の後にカンマを挿入することを忘れないでください。
>
> **label** オプションは、廃止された **taskName** オプションの後継です。
>
> 同じオプションラベルは、**launch.json** ファイルが起動タスクに関連する起動前タスクを見つける際にも使われるので、テキストは同じである必要があります。

- **label** オプションは、タスクのビルドの種類を反映するように変更されています。
- gcc コンパイラー命令 `-I${workspaceFolder}/include`. がインクルードされています。
- [エクスプローラー] ペインで選択されているファイルに応じてビルドが決定されないようになりました。つまり、任意のファイルをビルドできるため、ビルドに失敗する可能性があります。
- コンパイラーに、追加の検索フォルダーを知らせます。
- `${fileDirname}/${fileBasenameNoExtension}` を `${workspaceFolder}/bin/${config:programName}_d` に置き換えます。
- ビルド出力は bin フォルダーに生成されます。
- 実行ファイルの名前は、${config:programName} という変数で設定されます。これについては、後述します。
- **“isDefault”: true** ディレクティブが削除されました。

タスクから **isDefault** オプションを削除することで、ビルドするたびにユーザーがビルド構成を手動で選択することを Visual Studio* Code に知らせます。

これで、デバッグとリリースのビルドタスクの構成は完了です。コードをコンパイルできます。後で **Ctrl + Shift + b** キーを押すか、メニューから **[ターミナル] > [ビルド タスクの実行…]** を選択すると、図 5 に示すように、利用可能なビルドの種類がドロップダウン・メニューに表示されます。

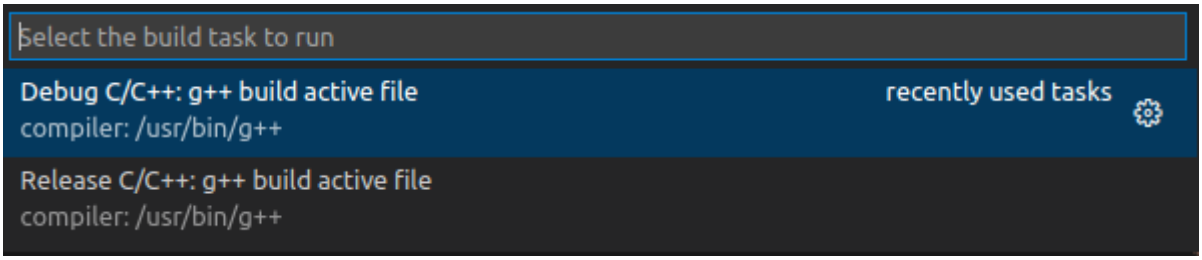

図 5: 利用可能なビルド実行ファイル構成のリスト

これらは、タスクの label オプションで設定されたビルドタスクの名前です。

このプロジェクトのリリースとデバッグのビルドタスクには、このほかにも以下の違いがあります。

- コンパイラーは、コンパイルシンボル **NDEBUG** を定義します。
- ファイル名からサフィックス **_d** が削除されています。

本記事の執筆時点では、図 6 に示すタスクオプション **env** が想定どおりに動作しないことが判明しています。代わりに、変数値を共有する別の方法を使用しています。

図 6: tasks.json ファイルのグローバル環境変数

グローバル環境変数を提供する別の方法として、Visual Studio* Code のプログラム設定を使用できます。.vscode フォルダーに **settings.json** ファイルを作成し、図 7 に示すように編集します。

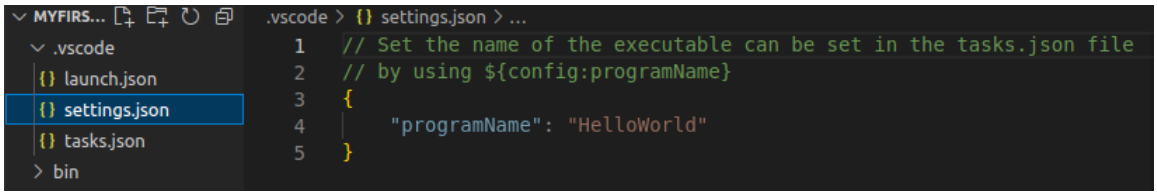

図 7: Visual Studio* Code の設定ファイル

tasks.json ファイル内のタスクで環境変数を使用するには、オプションの値に `${config:programName}` を挿入します。

> **注:**
> 残念ながら、いくつかのオプションのテキスト値では **${config:programName}** の置換が機能しません。例えば、タスクの **label** オプションでは機能しません。

これで、プロジェクトのビルド構成が定義されました。プログラムをコンパイルできます。**Ctrl + Shift + b** キーを押して、ビルドするプログラムの種類を選択します。Visual Studio* Code のターミナルウィンドウまたは外部のターミナルウィンドウを使用して、bin フォルダーにある実行ファイルを探します。

プロジェクトのデバッグ実行ファイル構成の設定

launch.json ファイルには、複数のデバッグセッション構成を含めることができます。このプロジェクトでは、起動構成は 1 つだけです。以下の手順に従って、このプロジェクトのデバッグ構成を作成します。

1. [エクスプローラー] ペインでメインの .cpp ファイルを選択します。このプロジェクトでは、HelloWorld.cpp ファイルを選択します。
2. Visual Studio* Code のメニューから **[表示] > [コマンド パレット...]** を選択して、コマンドパレットを開きます。
3. ボックスに「**C/C++: デバッグ構成の追加**」を入力します。
4. **[デバッグ構成の追加]** コマンドを選択して、ドロップダウン・リストからプログラムのビルドに使用したのと同じコンパイラーを選択します。
5. [エクスプローラー] ペインに表示された新しい launch.json ファイルを選択して、図 8 に示すデバッグ構成のように編集します。
6. ファイルを保存します。

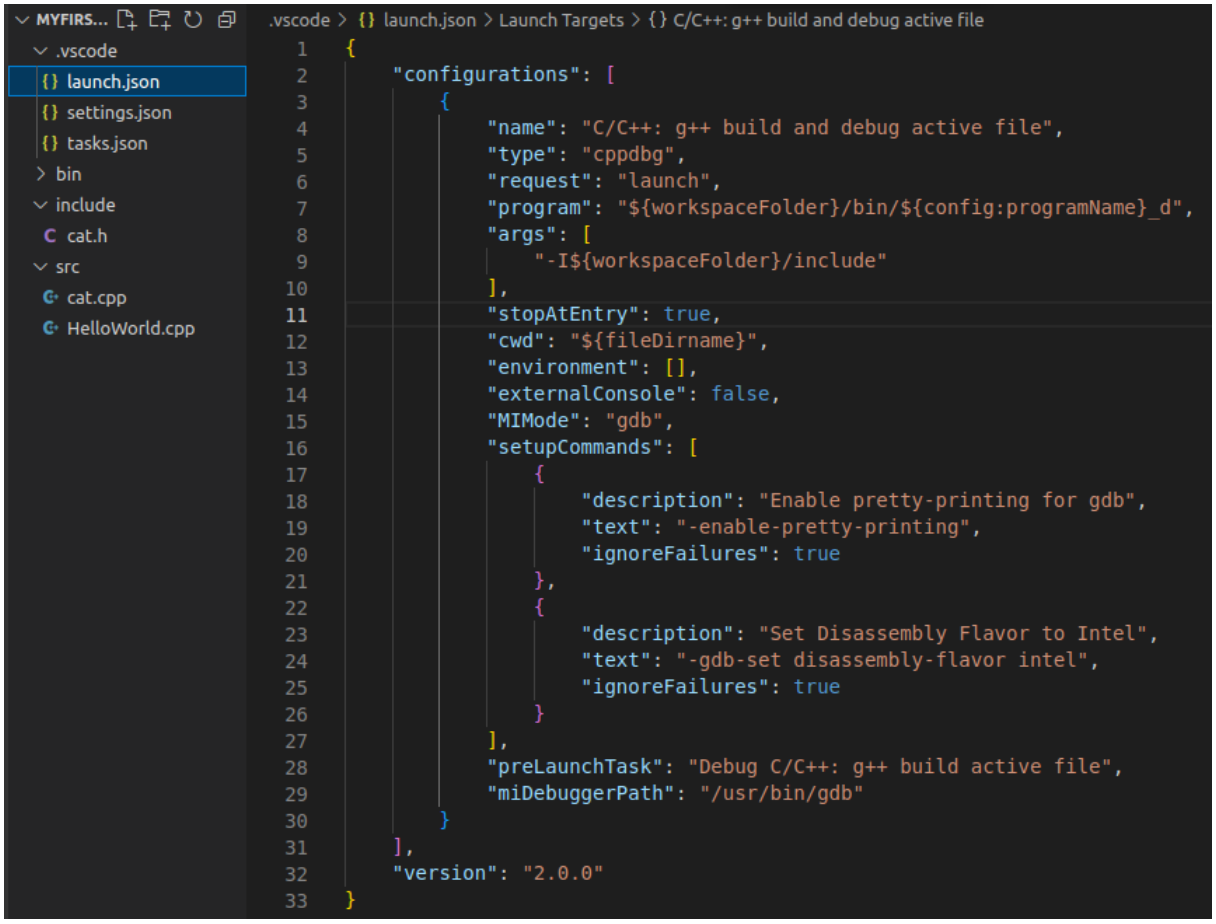

図 8: カスタマイズしたデバッグ起動構成

注:
デバッグセッション中に、デバッグ GUI パネルのシンボルの色が薄くなり、反応しなくなった場合は、デバッグセッションに失敗しています。新しいデバッグセッションで再スタートしてください。

必ず、独自の起動構成を作成して使用してください。Visual Studio* Code は、タスクが存在しない場合、タスクを自動生成してデバッグを続行します。あるいは、デフォルトで規定のタスクを使用したり、ビルドおよびデバッグオプションのリストを提供しますが、これらはすべてデバッグセッションを正常に起動しない場合があります。

構成タスクファイルで注目すべき変更点は、以下のとおりです。

- **program** オプションで環境変数の値を置換して、実行ファイルを識別するため同じ名前が使用されることを保証します。
- **stopAtEntry** オプションは true に 設定されています。
- **preLaunchTask** オプションは、tasks.json.on のデバッグ・ビルド・タスクのラベルと一致するように変更されています。
- **“isDefault”: true** オプションが削除され、異なるタイプのデバッグセッションをいつでも呼び出せるようになりました。

  注:
  "isDefault": true オプションを削除することで、デフォルトではなく、特定の起動設定を選択できます。

  起動前タスクが必要ない場合は、**preLaunchTask** オプションを削除します。

デバッグセッションを実行するには、F5 キーを押します。このれいでは、デバッグ・ビルド・タスクが実行され、実行ファイルがリビルドされます (変更されていなくても)。次に、GUI がデバッグモードに変わり、Helloworld プログラムコードが表示され、デバッガーはプログラムの `main()` スコープで停止されます。図 9 を参照してください。

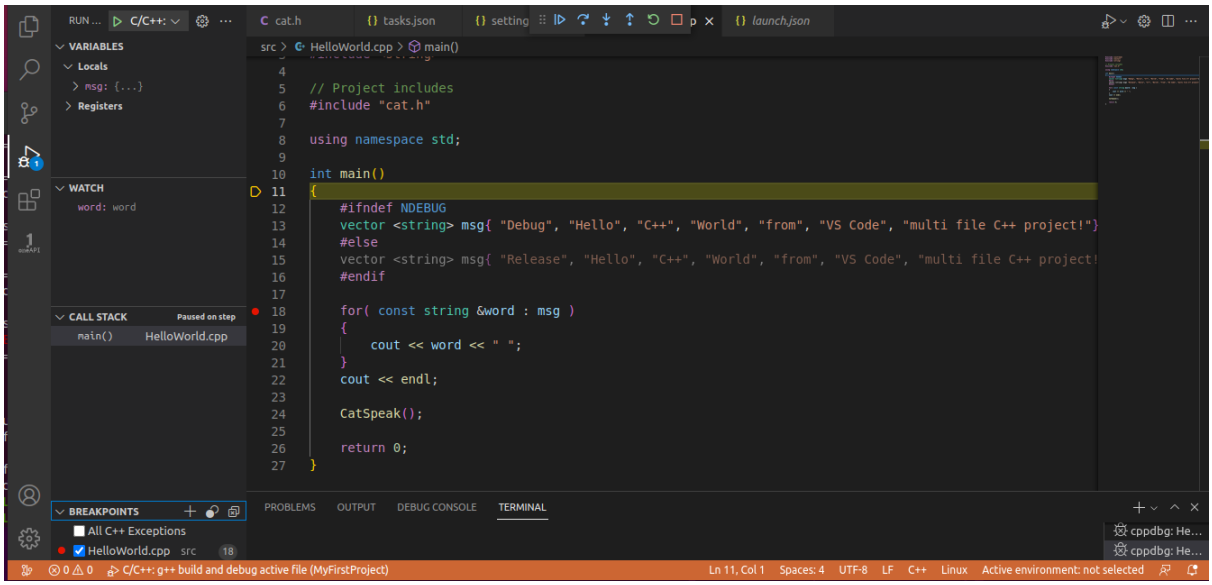

図 9: デバッグ中の Helloworld プログラム

Helloworld プログラムは、デバッグパネルまたはキーボード・ショートカットを使用して、途中でブレークポイントを設定したり、無効にしながら、ステップ実行できます。

このプロジェクト (フォルダー) を閉じて、再度開くと、前のセッションで設定したウォッチ変数やブレークポイントをすべて含んだ状態の同じデバッグセッションに戻るはずです。

これで、通常の C++ プログラム用のプラットフォーム上に、安定した信頼性の高い Visual Studio* Code の C++ 開発環境を構築し、ビルドおよびデバッグ・プロジェクトが動作することを確認できました。パート 3 では、インテル® oneAPI ツールキットとその依存関係を C++ 開発環境にインストールし、OS で低レベルシステムへのユーザーアクセスを適切に設定したら、このプロジェクトに戻り、通常の C++ プログラムをビルドしてデバッグできることを確認します。

SYCL* 開発向けに oneAPI、DPC++、Visual Studio* Code を設定

C++ 開発環境の準備ができたら、SYCL* 開発向けに oneAPI と DPC++ を設定します。

パート 3: Ubuntu* で SYCL* 開発向けに oneAPI、DPC++、Visual Studio* Code を設定
