Toradex
Linux C言語アプリケーション
開発マニュアル
BSP5用

本マニュアルは岡本無線電機株式会社が独自作成したものでありメーカーが保証した内容ではありません。万が一本マニュアルに間違いがあり、事故が生じたとしても岡本無線電機株式会社は一切の責任を問われないものとさせていただきます。

変更履歴

バージョン	更新日付	変更内容
1.00	2022/11/07	BSP5.7用に作成

本マニュアルについて

本マニュアルはトラデックスのCPU**モジュール上で動作する**C/C++**言語アプリケーションを作成する手順を記述しています。**

参考：
http://developer.toradex.com/knowledge-base/board-support-package/openembedded-(core)
https://developer.toradex.com/knowledge-base/linux-sdks

1．実行環境

本マニュアルの実行環境は下記です。

仮想化ソフト：VMWARE Player v15.5.7
Host OS： Windows 10 21H2
Guest OS：Ubuntu Desktop 20.04LTS 64bit(**英語版**)
BSP：v5.7
CPU**モジュール**：Verdin iMX8M Plus Quad 4GB Wi-Fi / Bluetooth IT V1.1A
キャリアボード： Verdin**開発ボード** Rev 1.1C + **アクセサリーキット**

本マニュアルとは異なるモジュールや評価ボード以外のキャリアボードを使われても大雑把には同じ操作となります。

インターネット接続環境が必要になります。

2. 事前準備

本マニュアルはLinux OSイメージ開発マニュアルの内容をすべて終えた状態で進めています。

3. 前提知識

Linux OSイメージ開発マニュアルの内容、TEZIによるOS書き込みをご理解いただいた状態を前提としています。

4. 注意点

オープンソース系を利用した開発に共通することですがすべてを理解しようとするときりがなく開発効率を損ないます。必要なタイミングで必要な知識を身につけるというスタンスで理解することを推奨いたします。

開発環境と実行環境の違いをわかりやすくするためにコマンドの表記の前に下記をつけています。

開発環境(パソコン)上で入力するコマンド：[Ubuntu]$

実行環境(モジュール)上で入力するコマンド(Linux) ：[Module]#

実行環境(モジュール)上で入力するコマンド(U-Boot)：[U-Boot]#

コピーについて

本マニュアル内のコマンドなどをコピーした場合、改行が入ったり「-」が抜けてしまうことがあるのでご注意ください。一度テキストエディタなどに張り付けてコピーした内容をご確認ください。

SDKの作成

C言語開発を行うためにOpen Embeddedでターゲットイメージ向けのSDKを作成します。
[Ubuntu]$ cd /work/oe-core

[Ubuntu]$. export

[Ubuntu]$ bitbake -c populate_sdk tdx-reference-multimedia-image

/work/oe-core/build/deploy/sdk配下にSDKが出力されます。

実行してSDKを展開します。(モジュールやBSPのバージョンによってシェルの名前が変わります。)
[Ubuntu]$ /work/oe-core/build/deploy/sdk/tdx-xwayland-glibc-x86_64-Reference-Multimedia-Image-aarch64-verdin-imx8mp-toolchain-5.7.0.sh

下記のようにSDKのインストールディレクトリを問われるので入力します。
[Ubuntu]$ Enter target directory for SDK (default: /opt/tdx-xwayland/5.7.0):
本マニュアルではデフォルトの/opt/tdx-xwayland/5.7.0にインストールしますのでそのままEnterキーを押します。

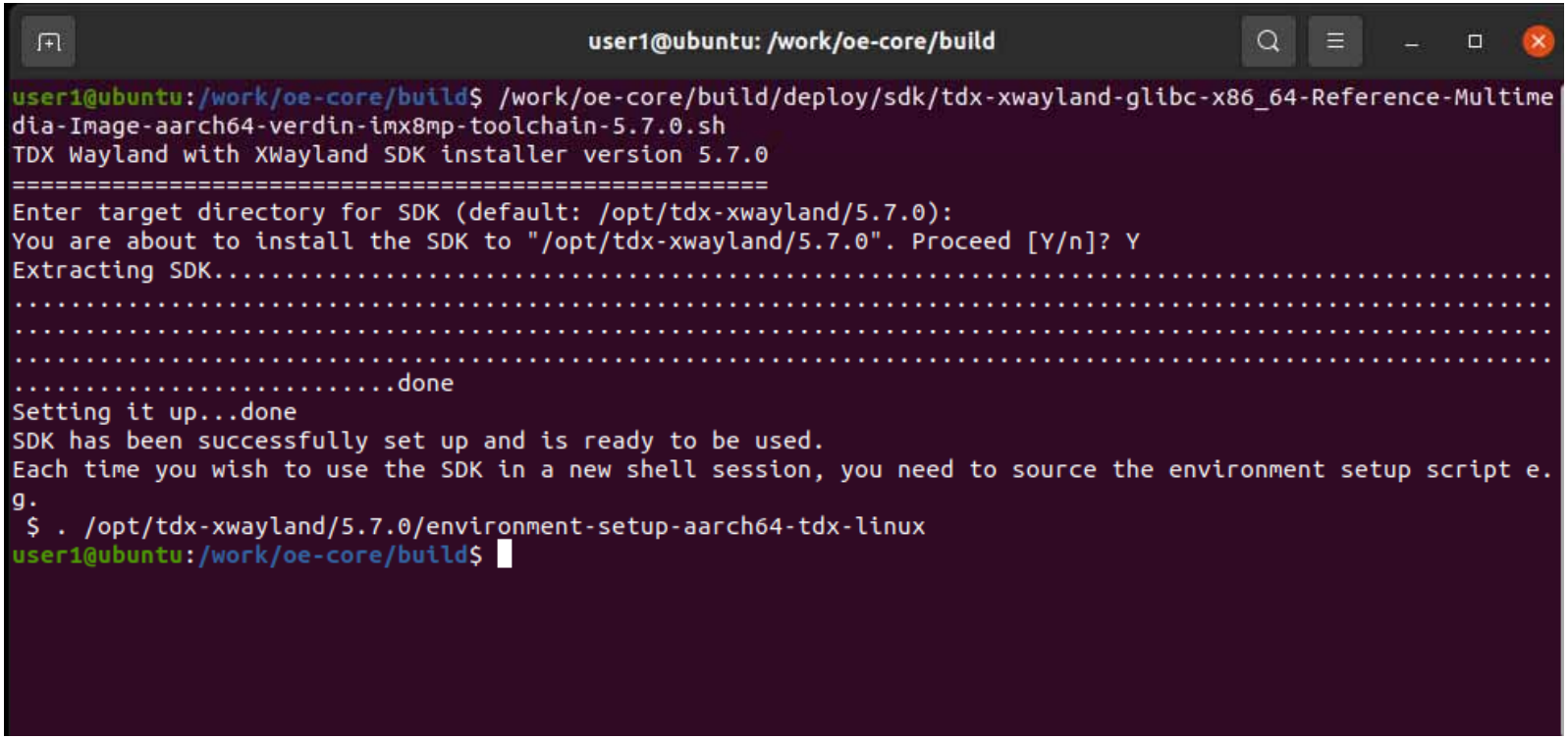
下記のように問われますので
[Ubuntu]$ You are about to install the SDK to "/opt/tdx-xwayland/5.7.0". Proceed [Y/n]?
Yを入力してEnterキーを押します。
管理者権限で実行しますので途中でパスワードを問われます。パスワードを入力してください。

すでに存在していた場合は下記のように上書きするかどうかを問われますがその場合は一度シェルを停止してシェル実行前にディレクトリを削除しておいたほうが良いです。
If you continue, existing files will be overwritten! Proceed[y/N]?

[Ubuntu]$ sudo rm -rf /opt/tdx-xwayland/5.7.0

最後に下記のような環境変数設定シェルのパスの案内があります。このシェルは後の工程で使用します。

. /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux



Eclipseのインストール

作業ディレクトリ作成
[Ubuntu]$ mkdir -p /work/app && cd /work/app

Eclipse**起動には**Java**が必要です。**Java**をインストールします。**
[Ubuntu]$ sudo apt-get -y install openjdk-11-jre

Eclipse (202006**バージョン**)**のサイトからを入手します。** (**ブラウザなどで入手**)
https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2020-06/R/eclipse-cpp-2020-06-R-linux-gtk-x86_64.tar.gz

app**配下にコピーします。**

展開します。
[Ubuntu]$ tar -xf ./eclipse-cpp-2020-06-R-linux-gtk-x86_64.tar.gz

Eclipse実行シェル作成

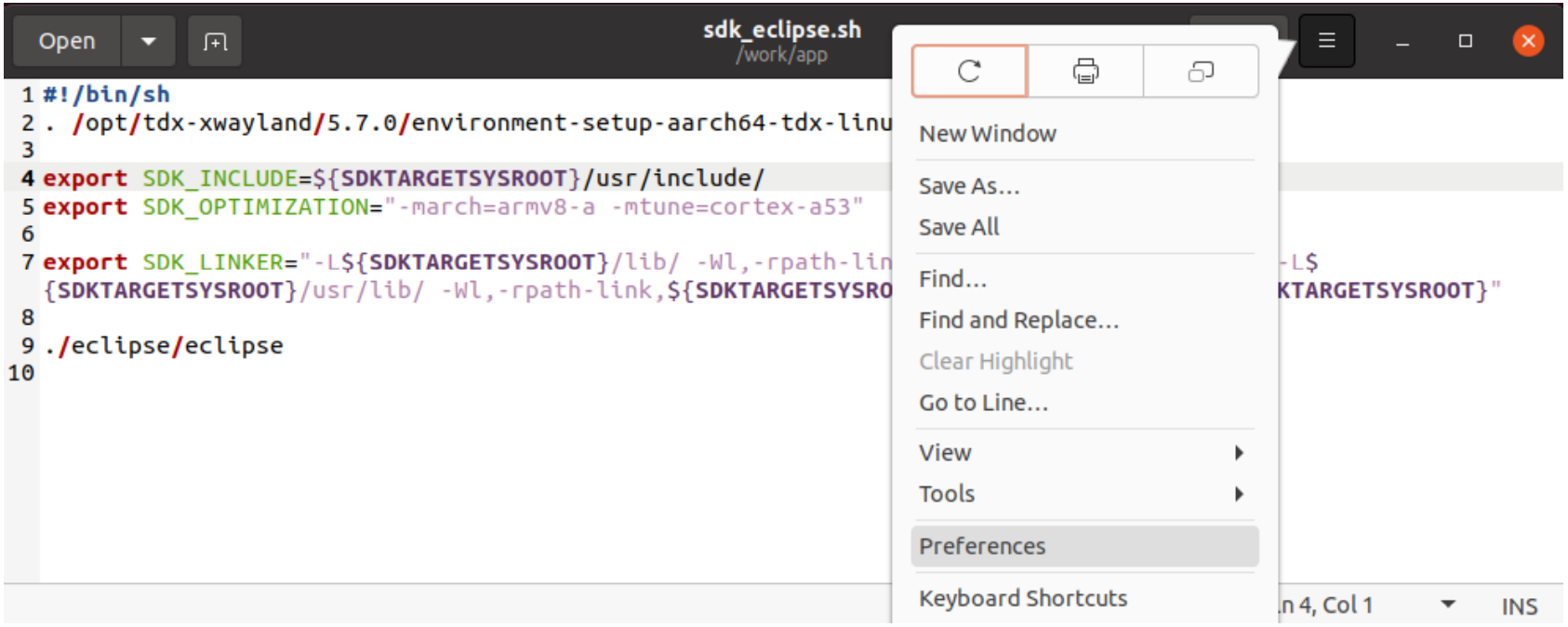
Eclipse起動前にSDK用環境変数を定義する必要があります。またコンパイルオプションも長くなるためSDK用環境変数設定やコンパイルオプションを定義してからEclipseを実行するシェルを作成します。
シェル内にはSDKが出力した環境変数設定シェルの実行も行っています。(長い赤線部分)
定義する内容は搭載するARMのアーキテクチャやBSPのバージョン、SDK出力ディレクトリなどによって変わります。

[Ubuntu]$ cd /work/app/
[Ubuntu]$ gedit ./sdk_eclipse.sh

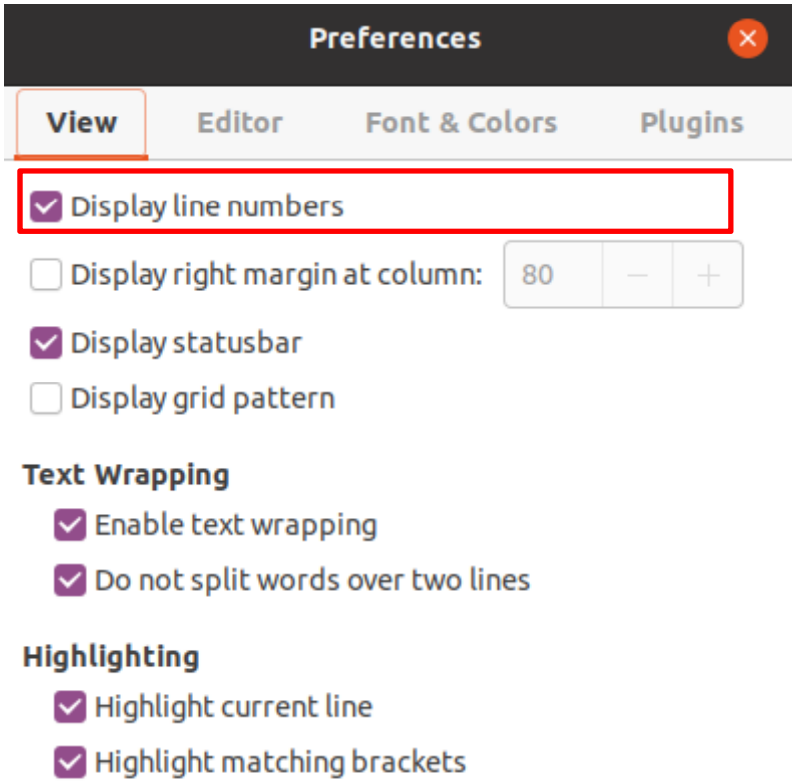
最適化オプションはCPUごとに指定を変える必要があります。下記に参考情報があります。
https://gcc.gnu.org/onlinedocs/gcc-9.3.0/gcc/AArch64-Options.html#AArch64-Options
https://ja.wikipedia.org/wiki/ARM%E3%82%A2%E3%83%BC%E3%82%AD%E3%83%86%E3%82%AF%E3%83%81%E3%83%A3
内容は下記です。

```
#!/bin/sh

. /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux

${SDK_ENV_SET}
export SDK_INCLUDE=${SDKTARGETSYSROOT}/usr/include/
export SDK_OPTIMIZATION="-march=armv8-a -mtune=cortex-a53"

export SDK_LINKER="-L${SDKTARGETSYSROOT}/lib/
 -Wl,-rpath-link,${SDKTARGETSYSROOT}/lib/ -L${SDKTARGETSYSROOT}/usr/lib/
 -Wl,-rpath-link,${SDKTARGETSYSROOT}/usr/lib/ --sysroot=${SDKTARGETSYSROOT}"

./eclipse/eclipse
```

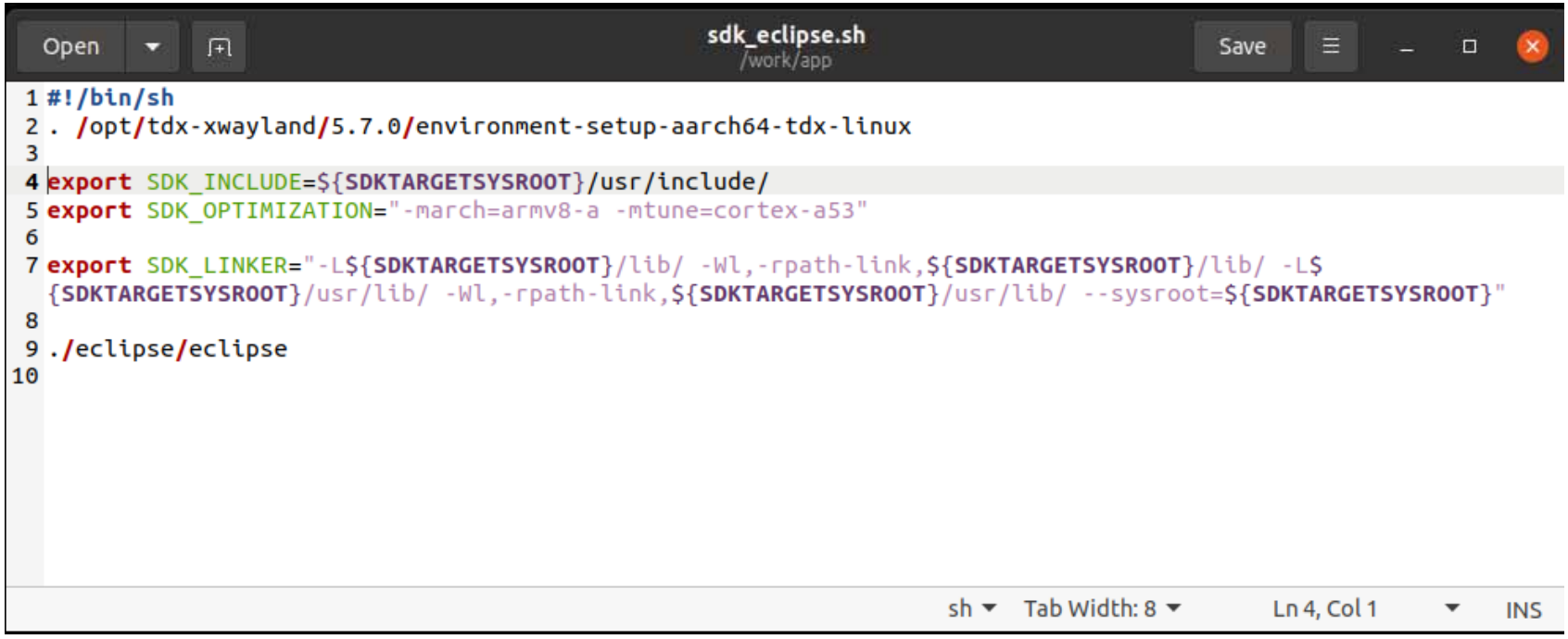
各々の設定が改行されていないかを確認してください。改行があるとうまくいきません。
geditで行番号を出す設定にするとわかりやすくなります。行番号を表示するにはgedit起動後画面右上のText EditorでPreferencesを開きます。



Display Line Numbersにチェックをいれて閉じます。



行番号が表示され各exportが一行で記述されているかどうかが分かりやすくなります。



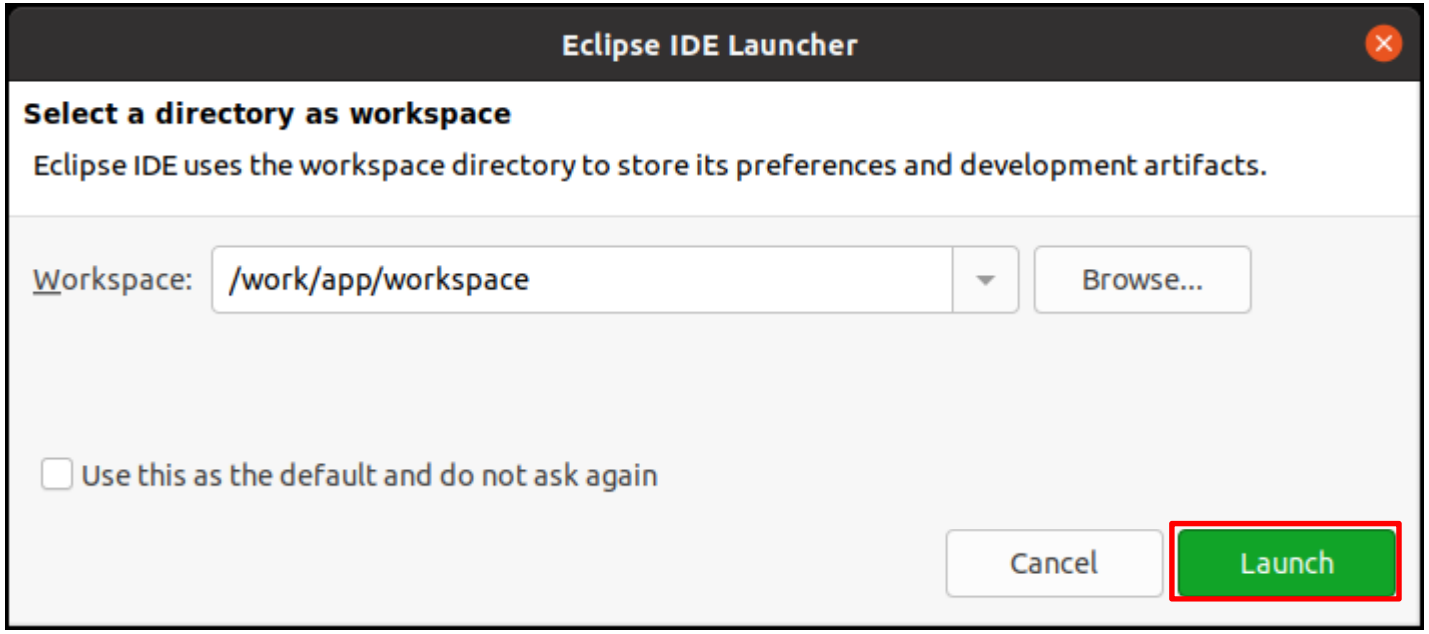
作成したファイルに実行権限を付与します。
[Ubuntu]$ chmod +x ./sdk_eclipse.sh

ワークスペースディレクトリ作成
[Ubuntu]$ mkdir ./workspace

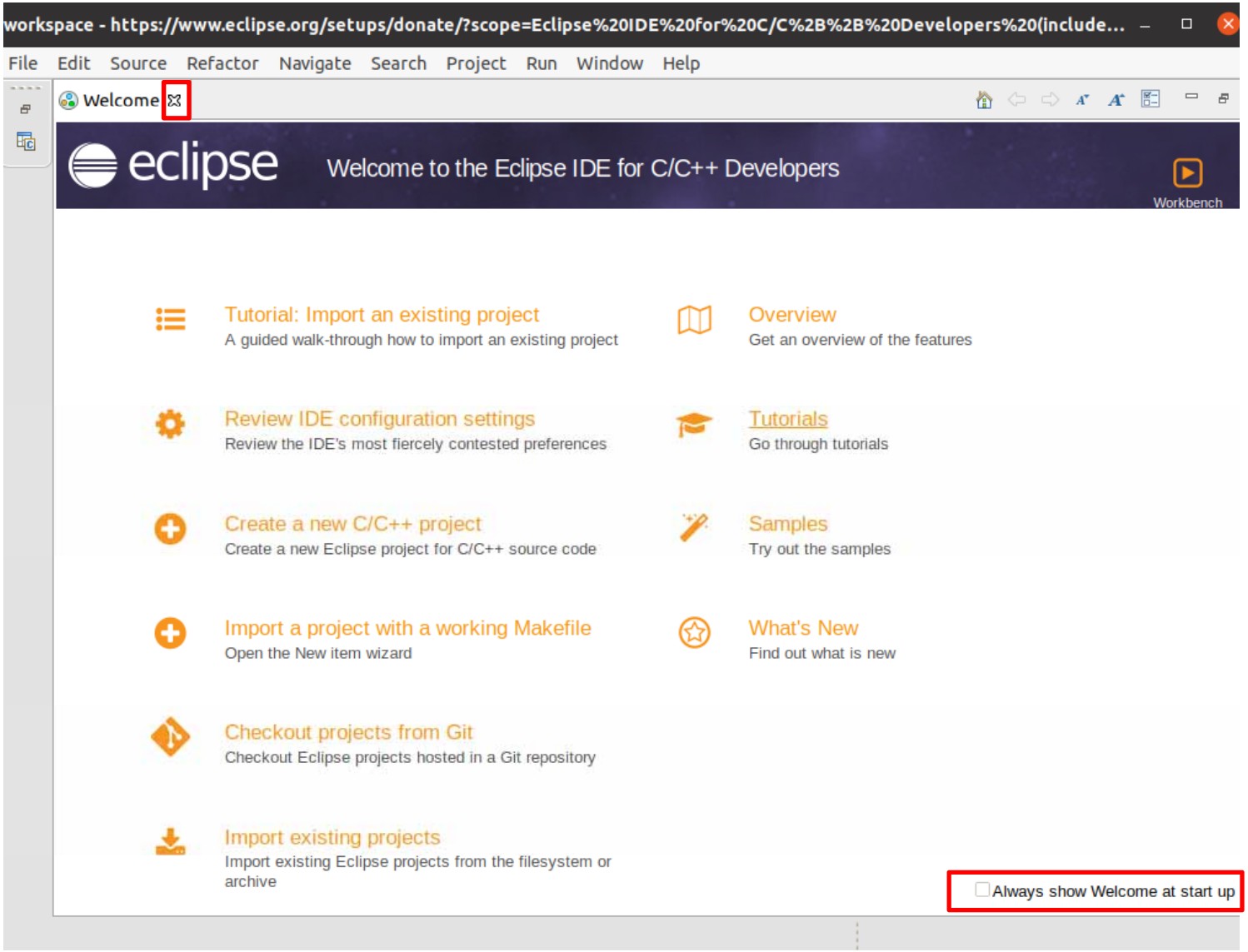
Eclipse**実行**
[Ubuntu]$ ./sdk_eclipse.sh

以後Eclipse**を起動する場合はこのシェルを使ってください。**

Eclipseを実行すると下記のようなワークスペースのディレクトリのパスを指定するウィンドウが表示されますのでワークスペースのパスを設定します。本マニュアルでは/work/app/workspaceにしています。
Launchをクリックしてを起動します。

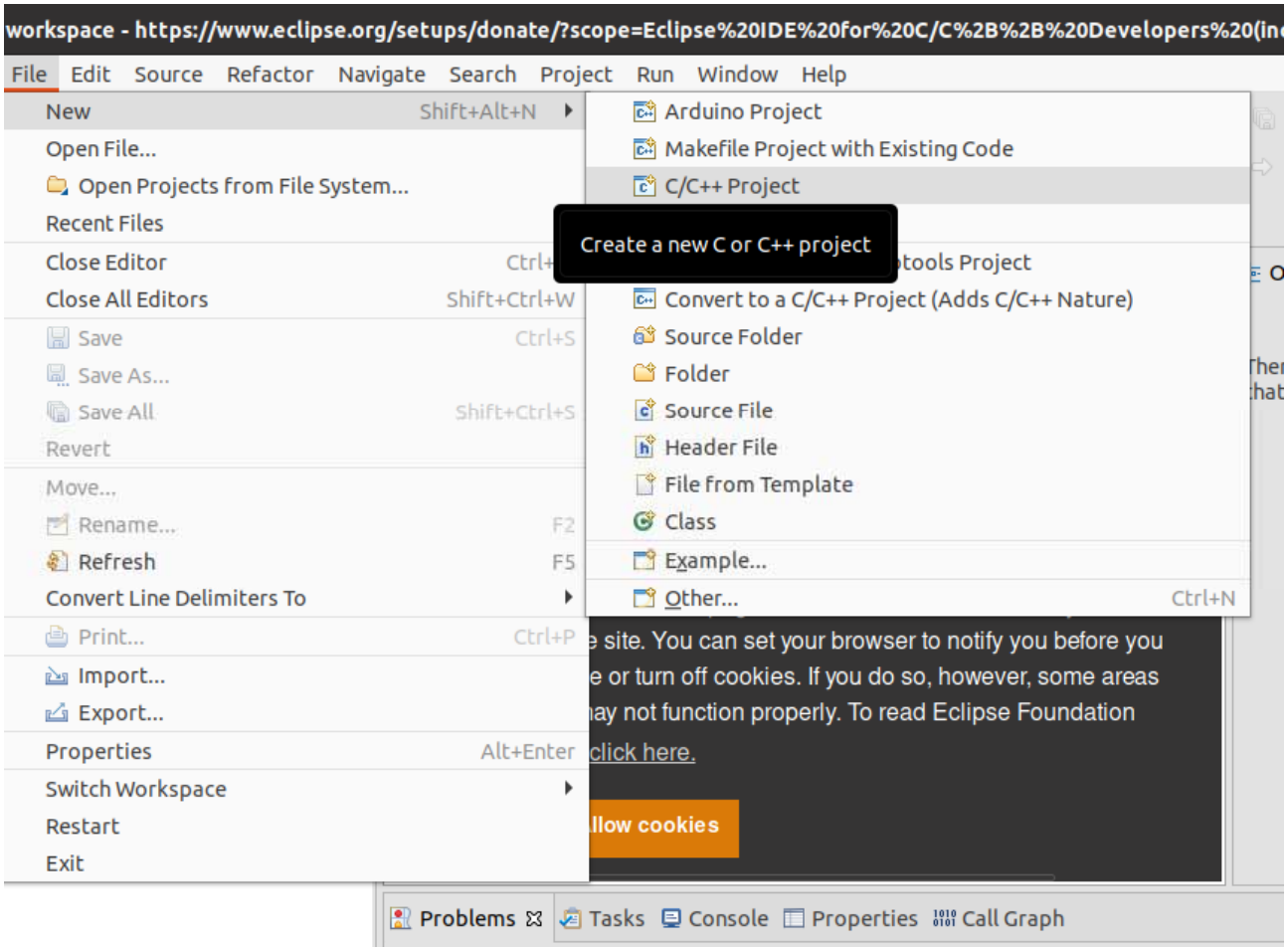


Welcomeと表示されますが必要ないので×ボタンをクリックして消します。
右下のAlways show Welcome at start upのチェックをはずいておけば毎回起動時に表示されることがなくなります。

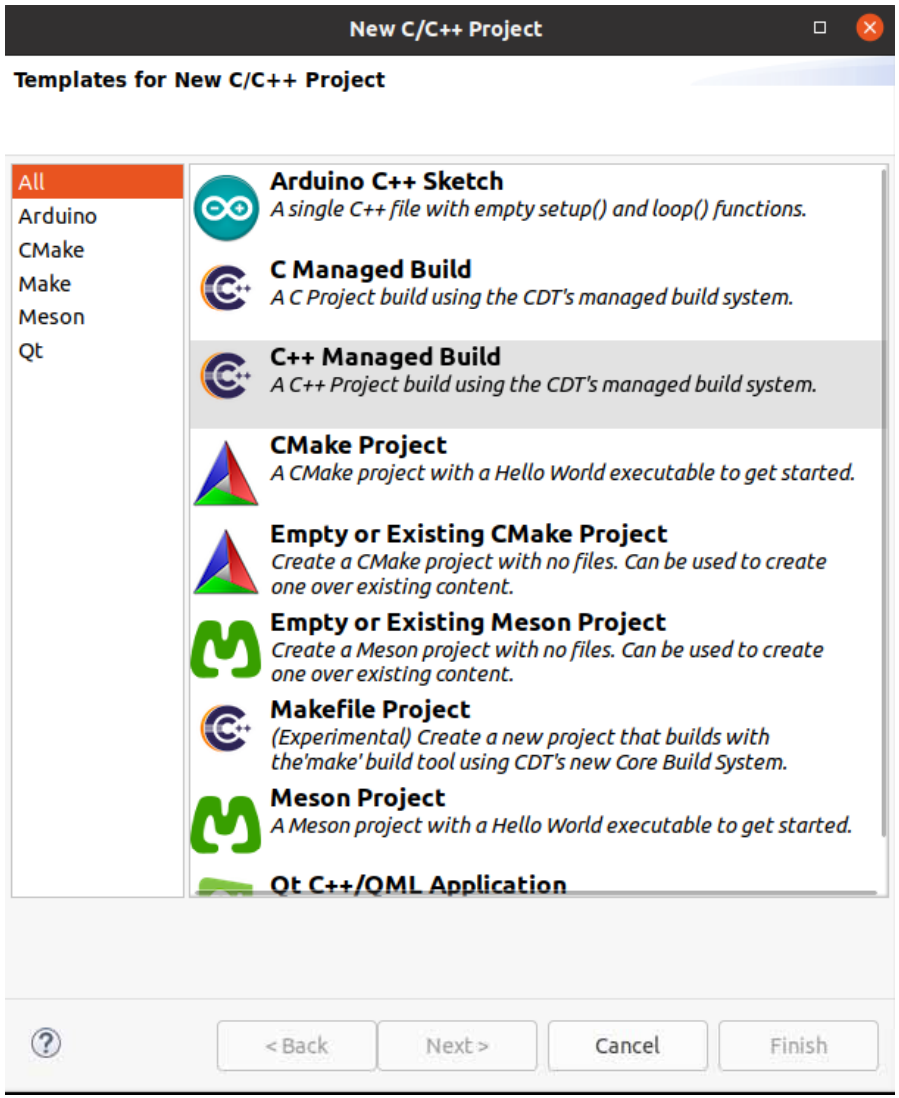


プロジェクトの作成＆ビルド

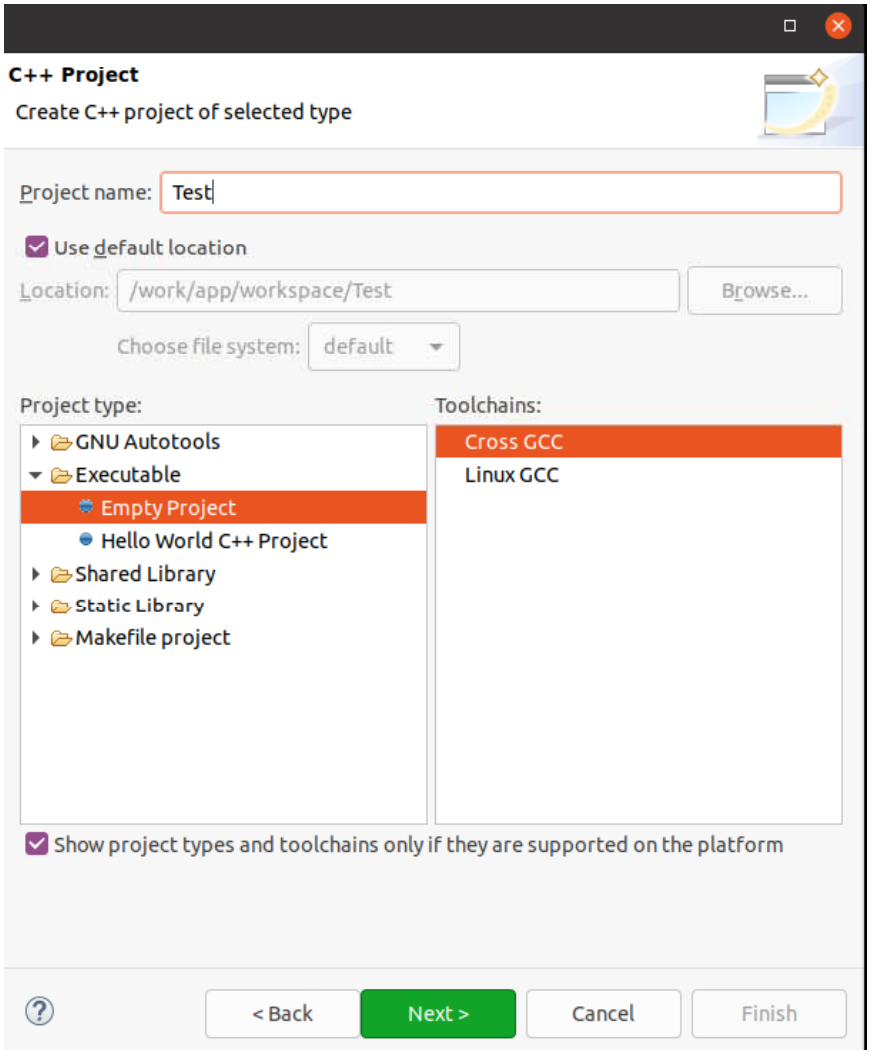
メニューからFile > New > C++Projectを選択します。



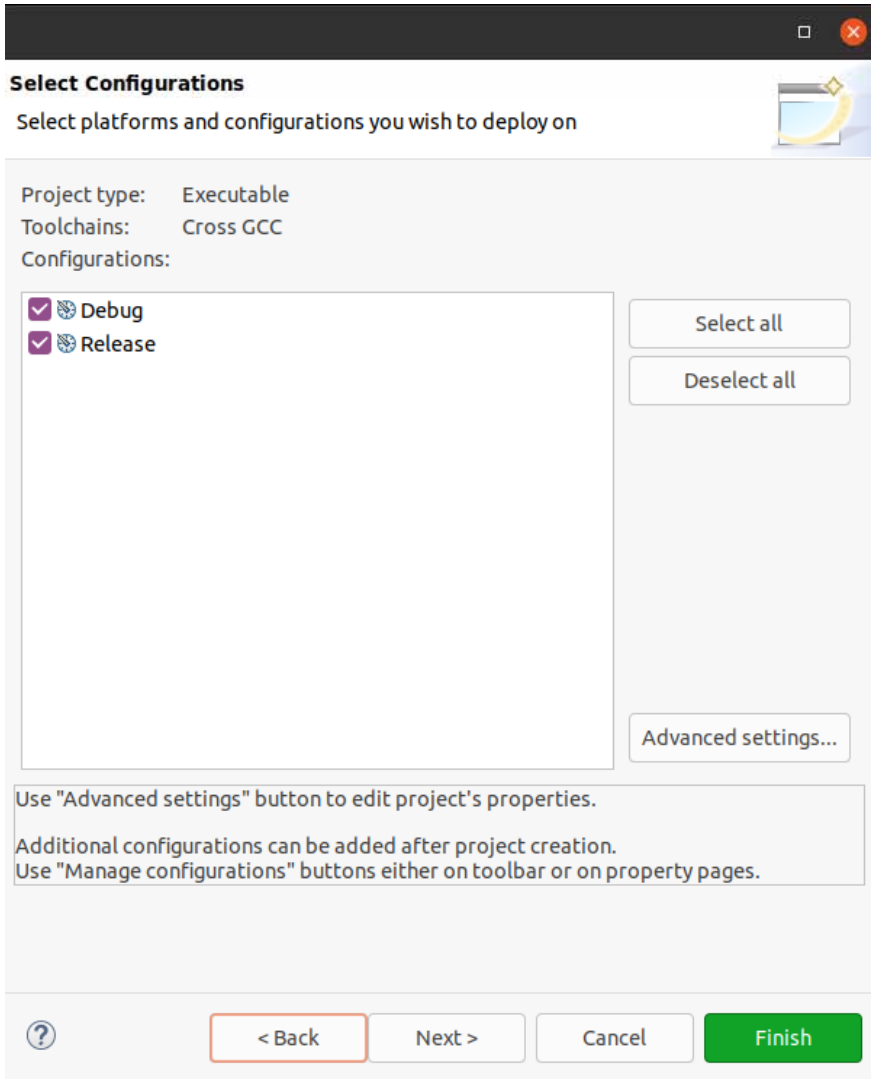
CもしくはC++言語のプログラムを作成するため「C++ Managed Build]を選択してNextをクリックします。



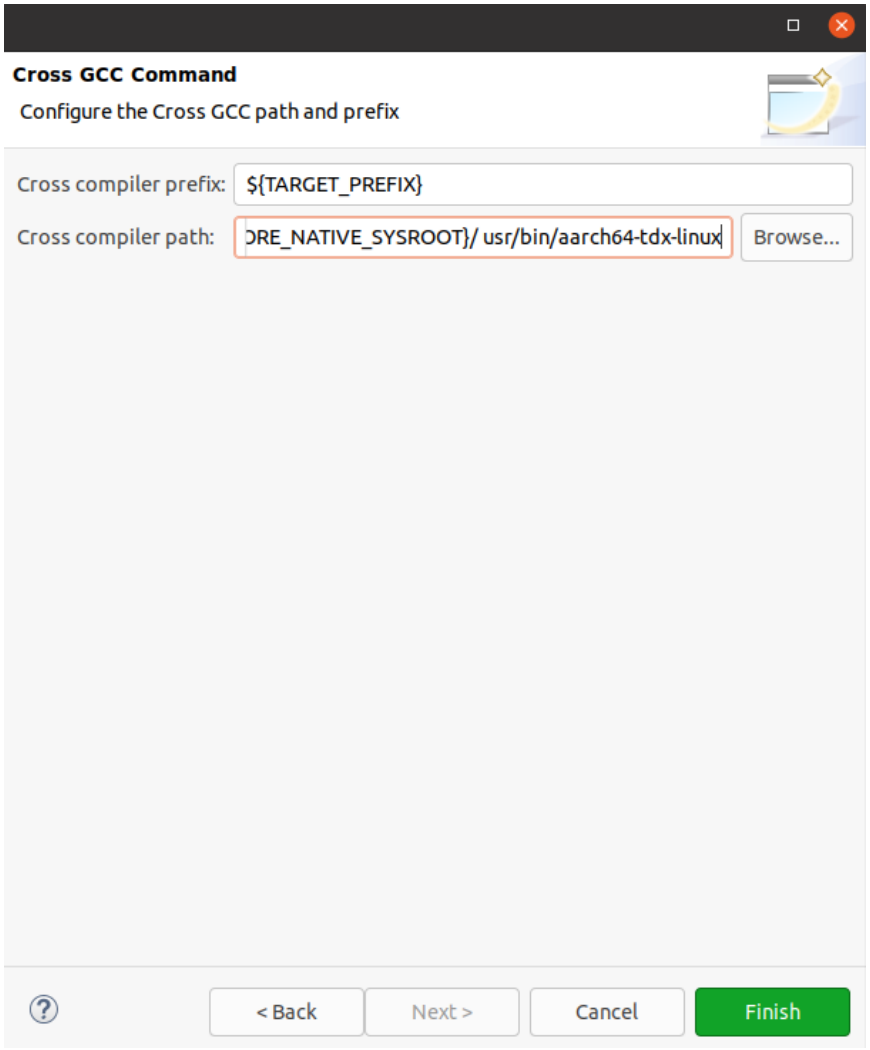
Project Nameを入力します。本マニュアルではTestにしています。
Project typeにEmpty Project、ToolchainsにCross GCCを選択します。
Nextをクリックします。



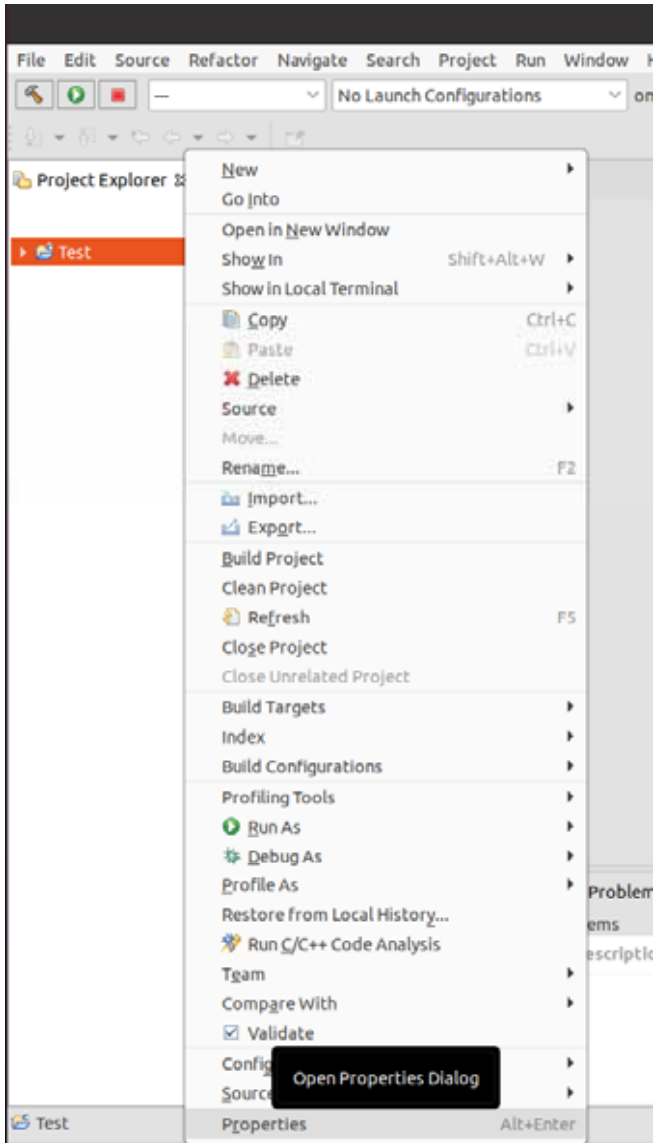
ここではデフォルト設定のままにします。DebugとRelease設定をもつProjectになります。
Nextをクリックします。



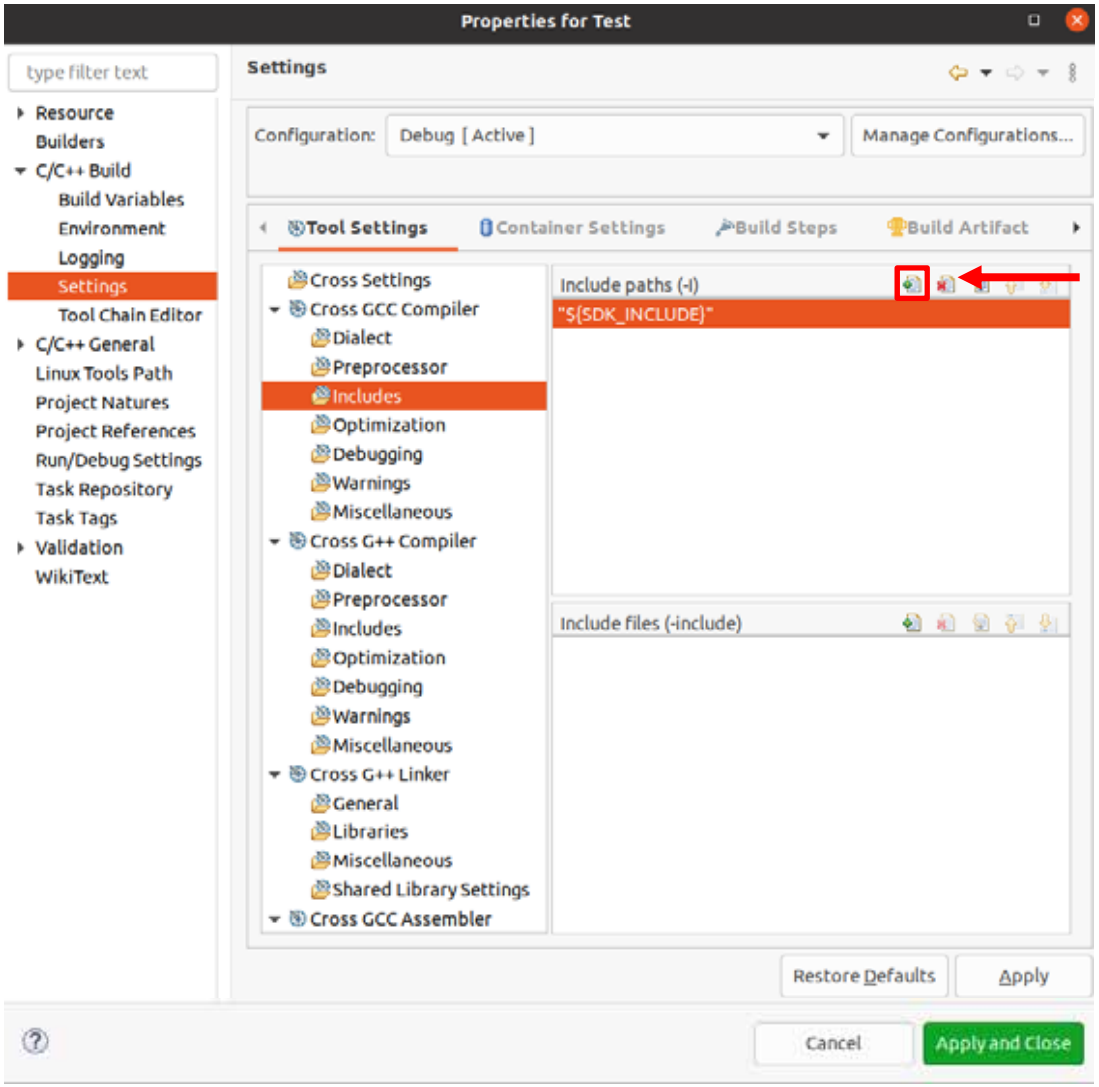
Cross Compiler prefixに「${TARGET_PREFIX}」
Cross Compiler pathに「${OECORE_NATIVE_SYSROOT}/ usr/bin/aarch64-tdx-linux」を入力してFinishをクリックします。
画面を大きくしないとCross Compiler pathの設定がすべて見えないので画面を最大化して確認してください。



作成したプロジェクトを右クリックしてPropertiesを開きます。

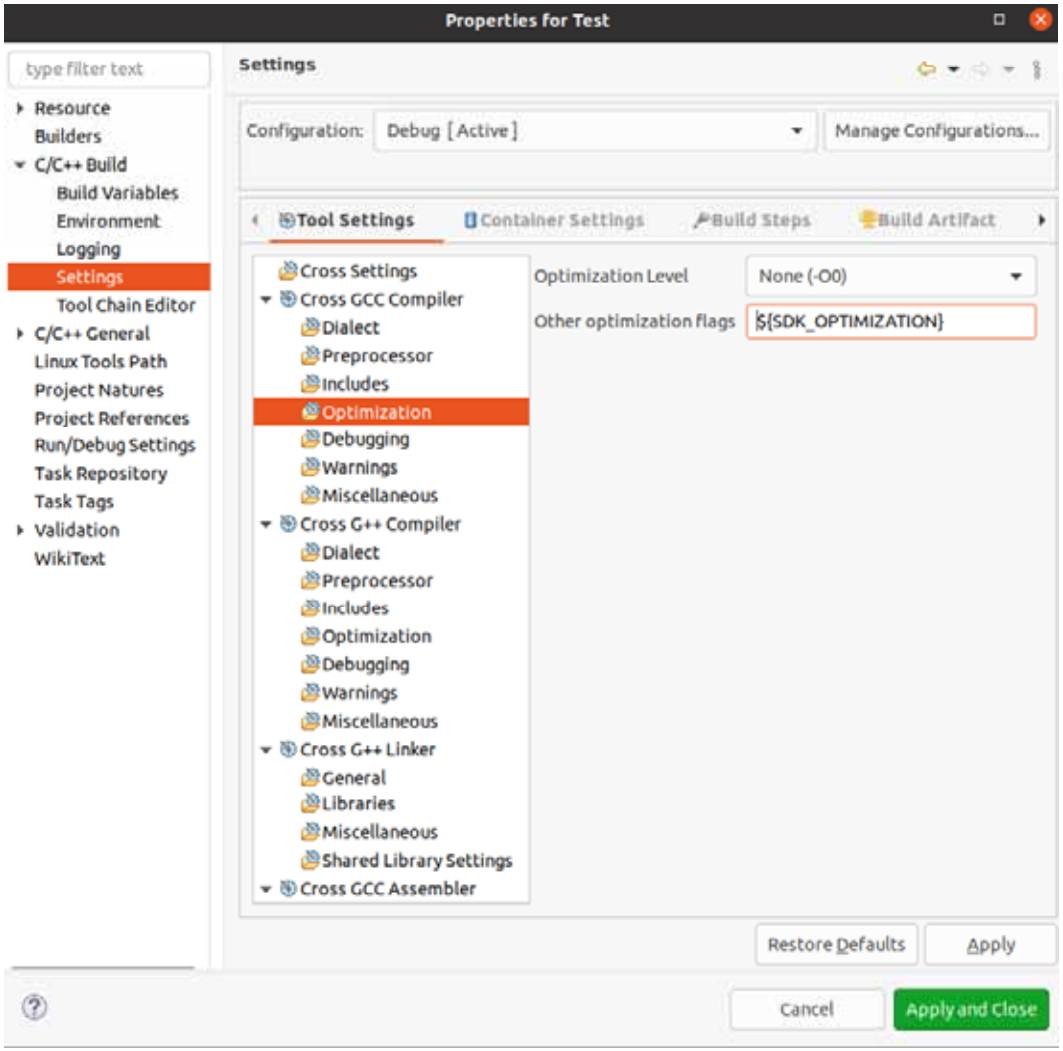


C/C++ Build > Settings > Cross GCC Compiler > Includesで+のアイコンをクリックして下記を設定します。
${SDK_INCLUDE}

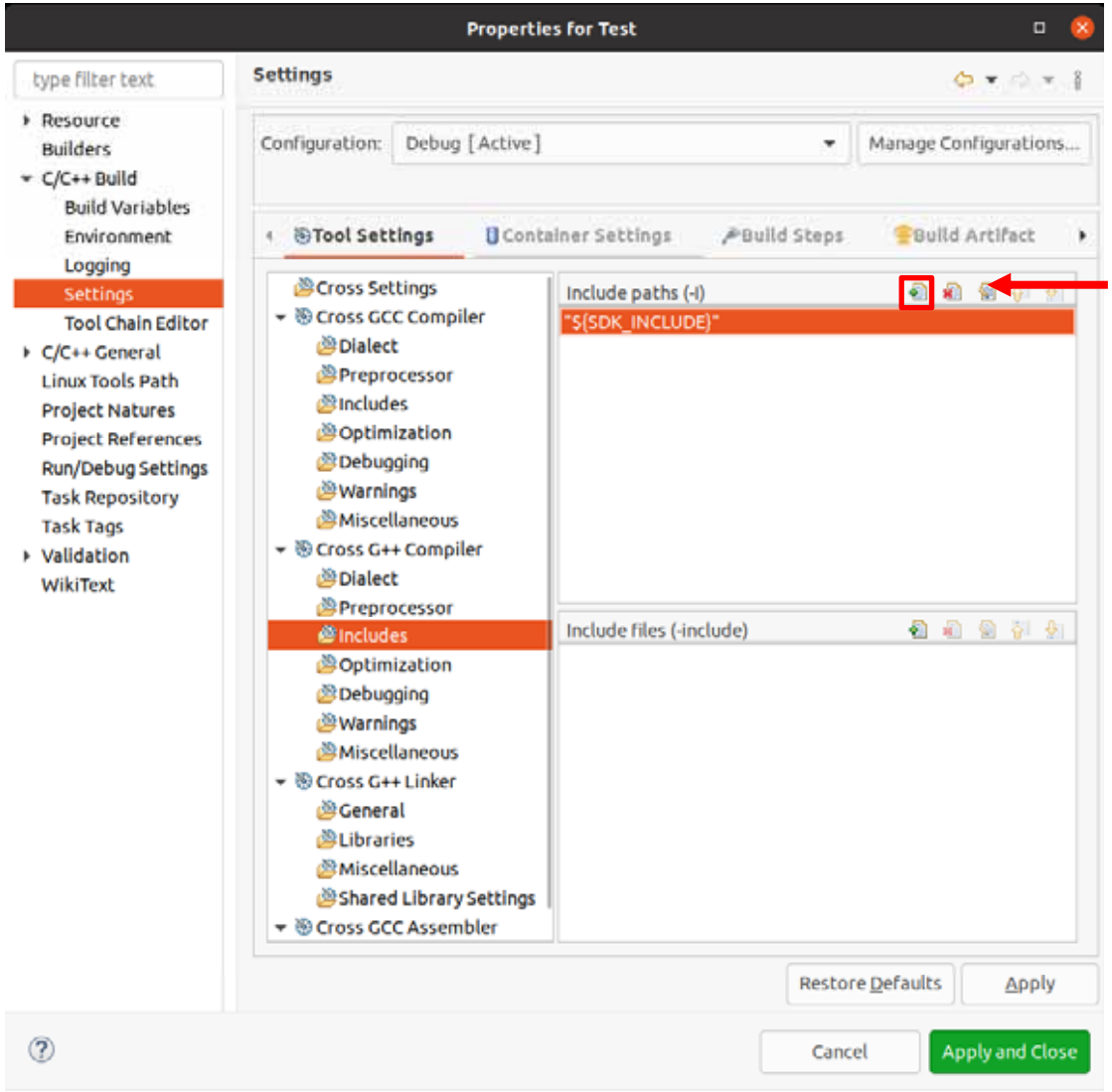


設定追加のアイコン

同画面のままOptimization**を選択し**Other optimization flags**に下記を入力します。**
${SDK_OPTIMIZATION}

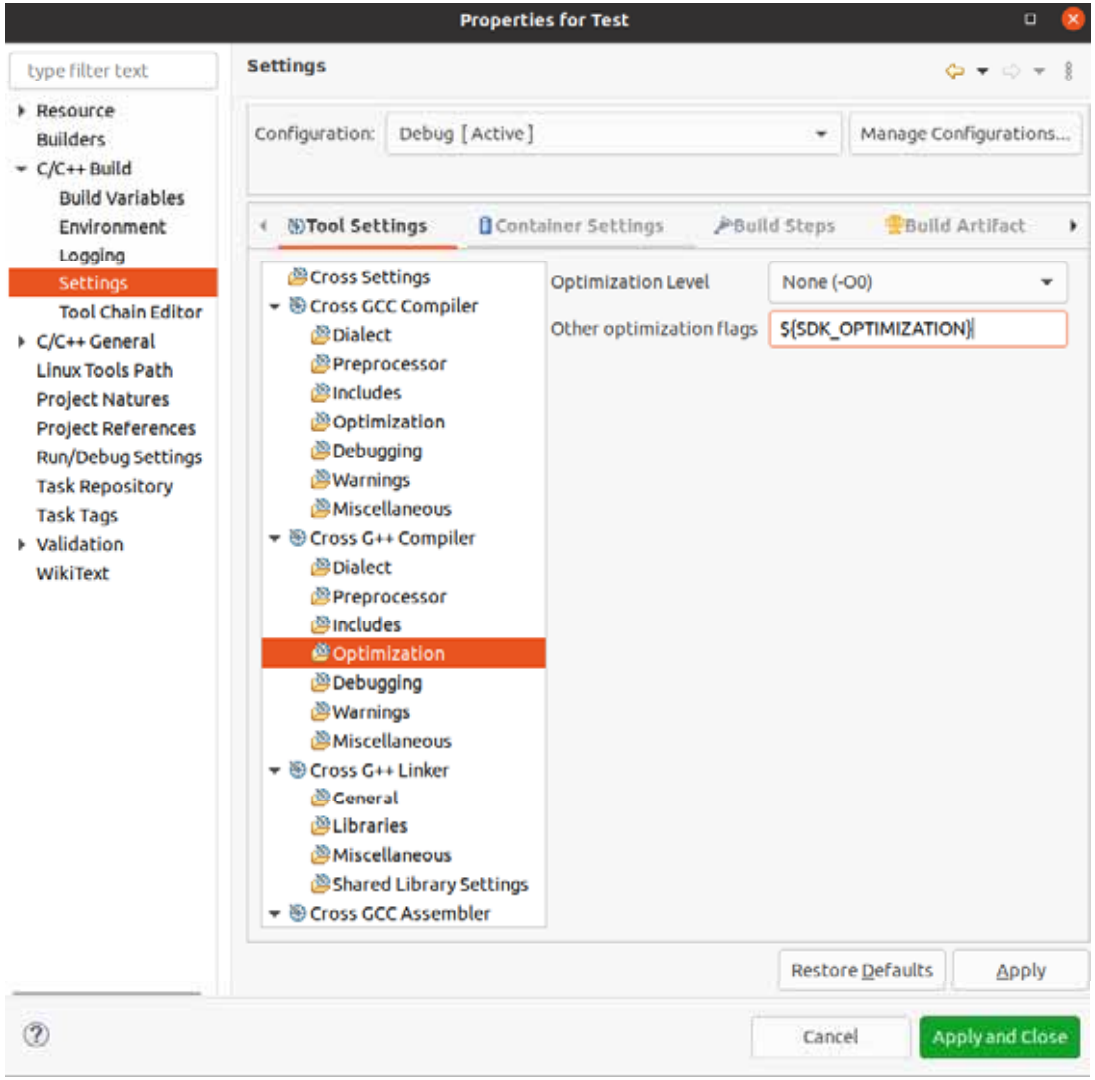


同画面のままCross G++ Compiler > Includes**にも同様に下記を設定します。**
設定を追加しただけで前回入力したものが入力されていますのでその場合は入力の必要はありません。
${SDK_INCLUDE}

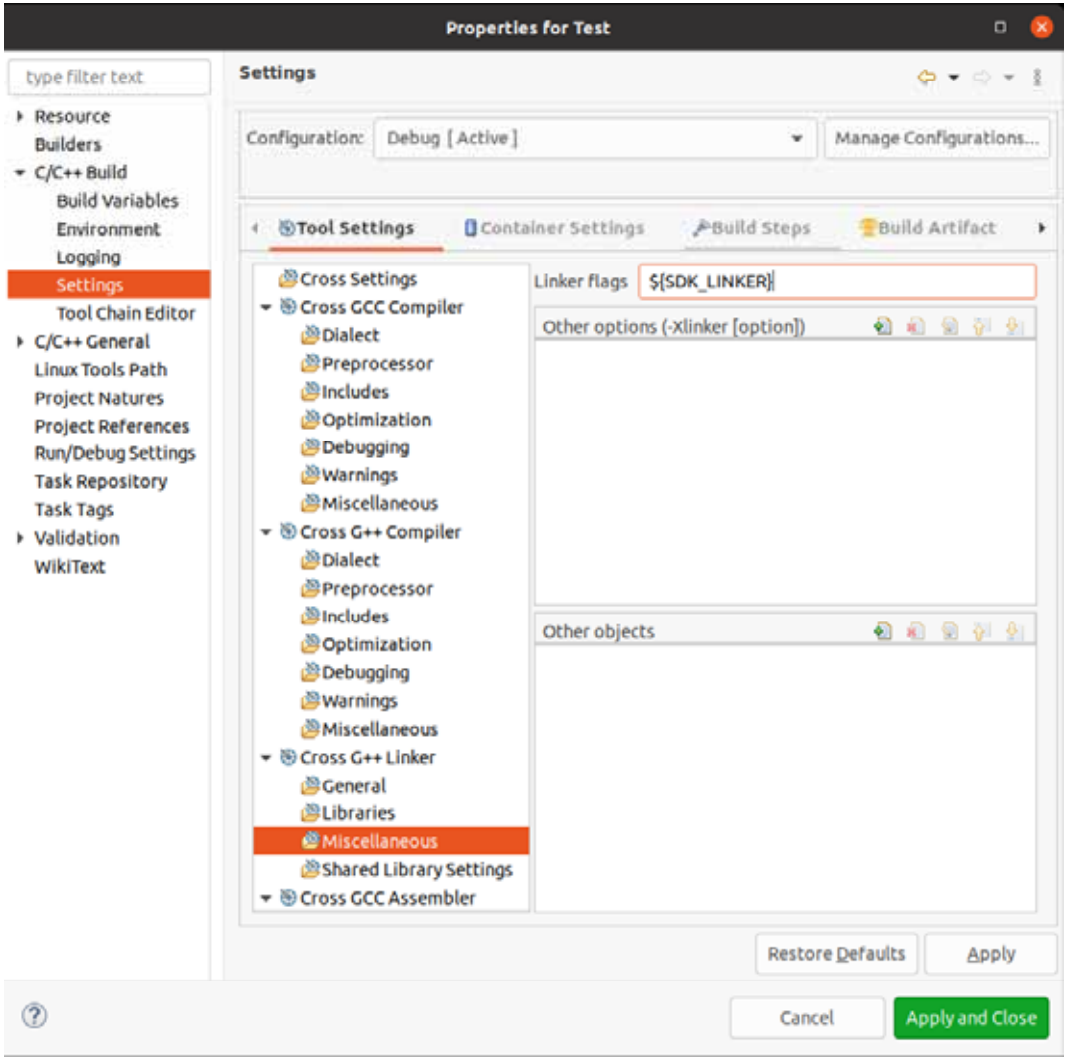


設定追加のアイコン

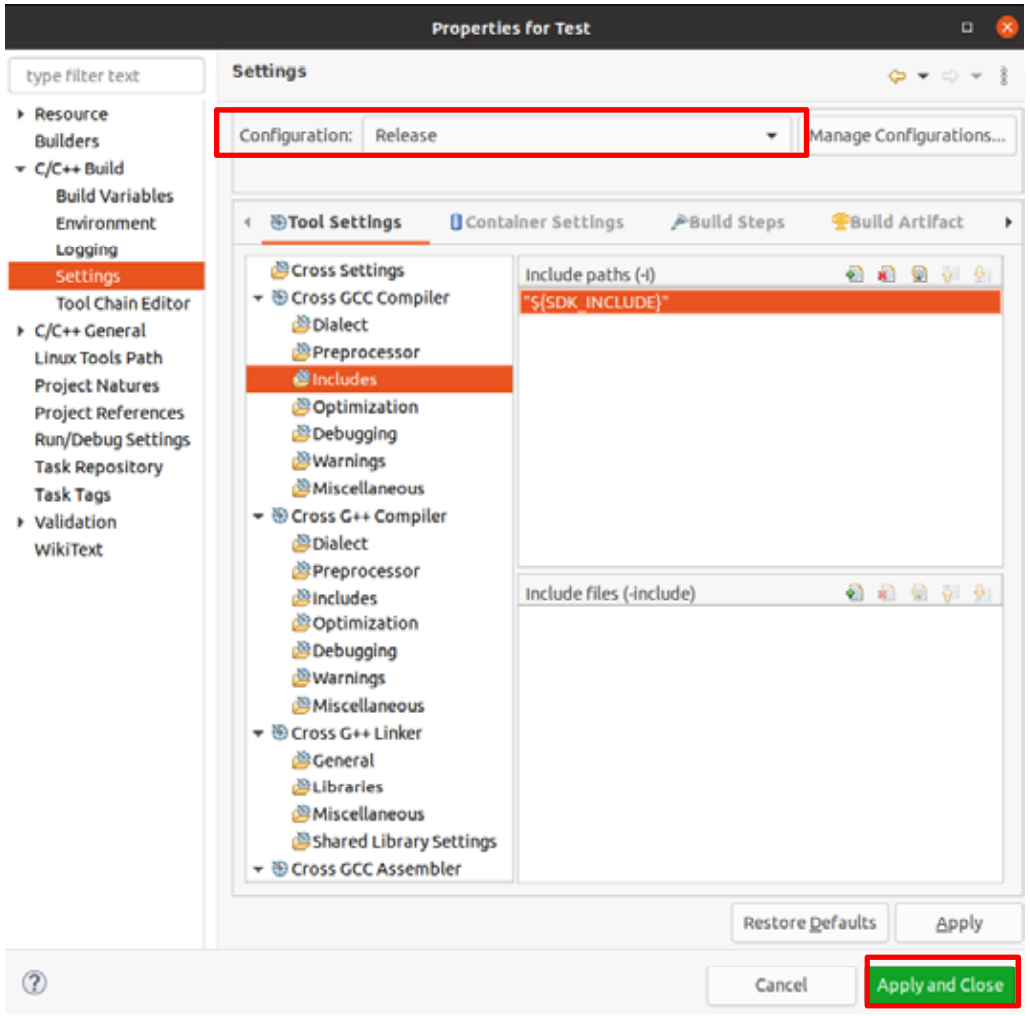
同画面のままOptimization**を選択し**Other optimization flags**に下記を入力します。**
${SDK_OPTIMIZATION}



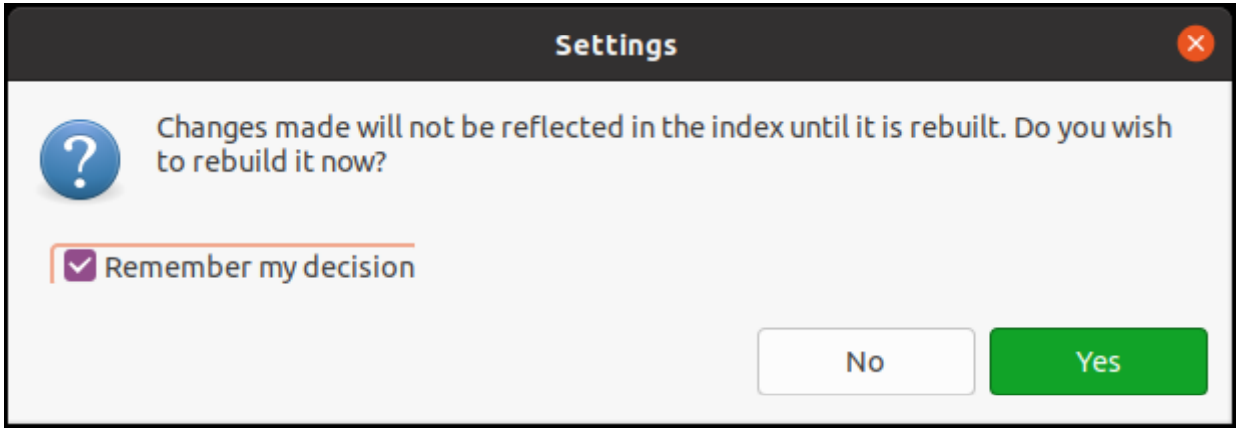
同画面のままCross G++ Linker > Miscellaneous**の**Linker flags**に下記を設定します。**
${SDK_LINKER}



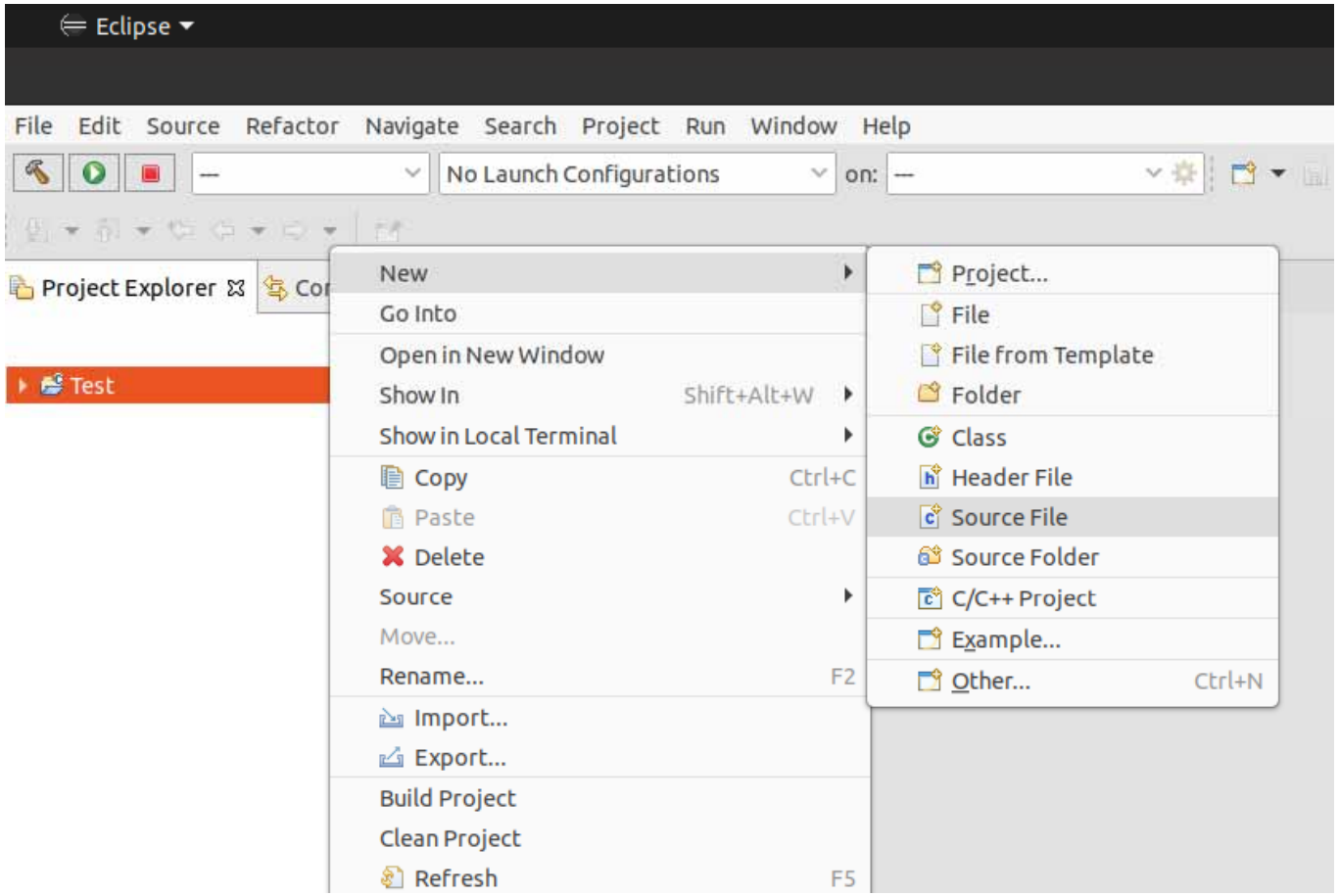
ConfigurationをReleaseにしてReleaseの設定にもIncludesやOptimization、Miscellaneousに同様のパラメータを入力します。
すべての設定入力後Apply and Closeをクリックします。



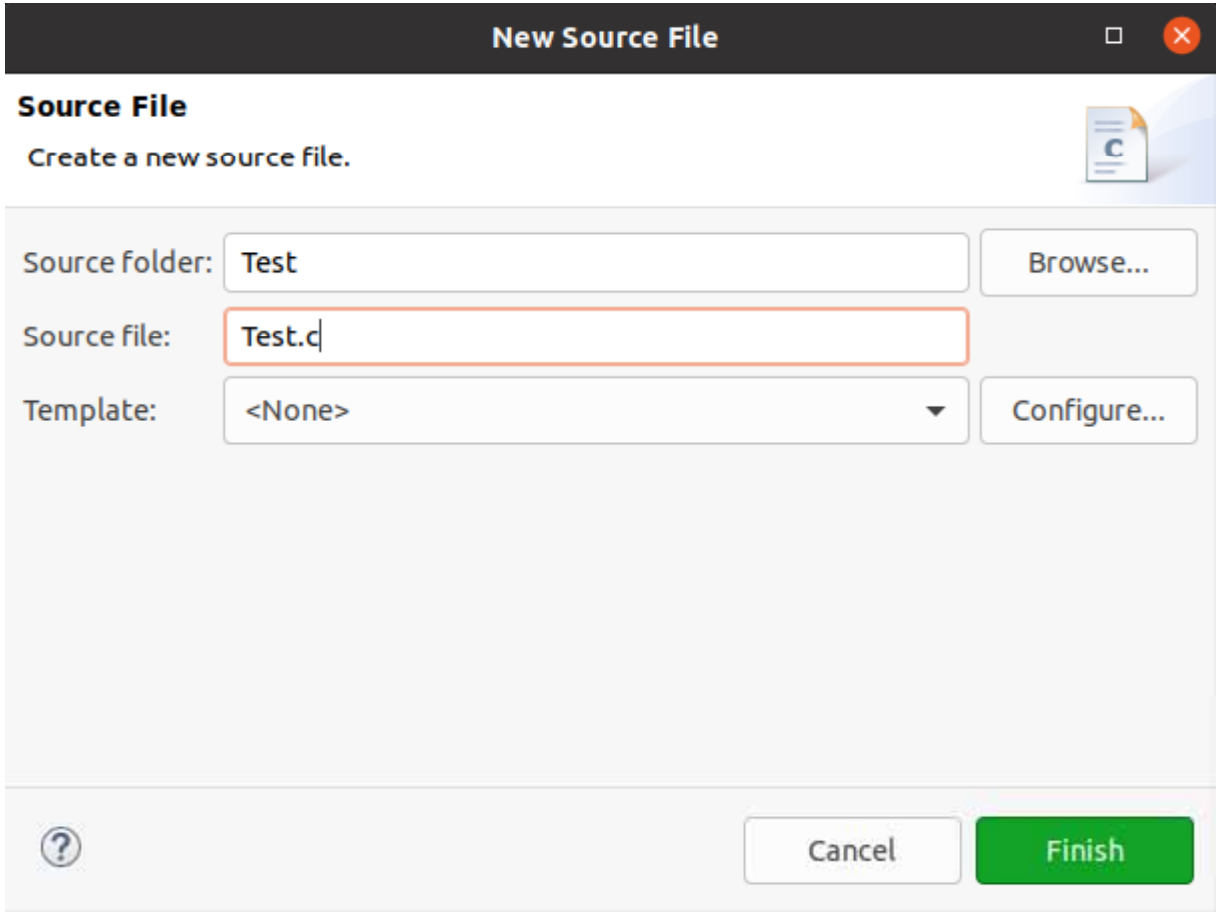
下記のようなリビルドするまで変更が反映されないという警告が出ます。
基本的にプロジェクトの設定変更後はリビルドして反映させますのでRemember my decisionにチェックをいれてYesをクリックします。



プロジェクトを右クリックしNew > Source Fileをクリックしてソースファイルを追加します。

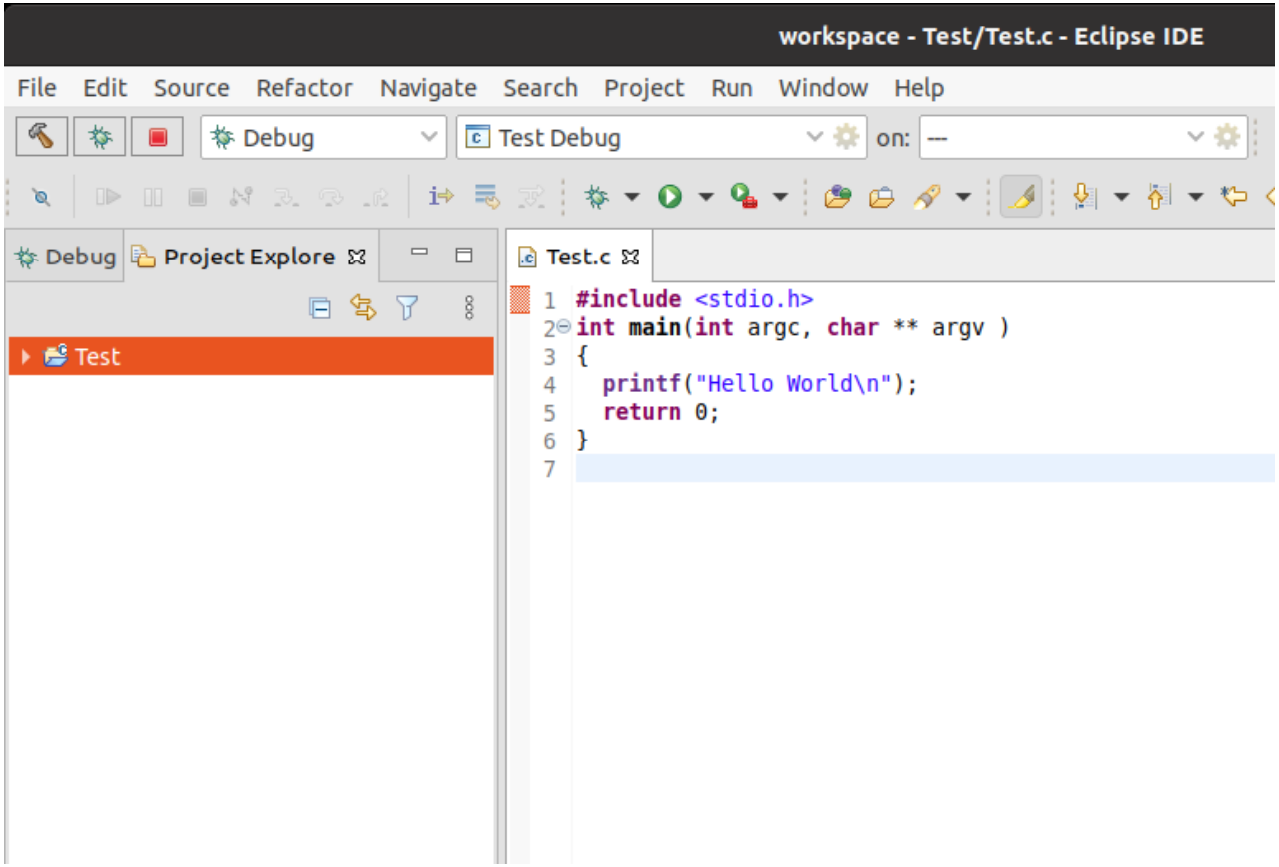


Source fileを入力します。拡張子を必ずつけてください。
Templateはソースコードに付加されるコメントですが本マニュアルではNoneとしています。Finishをクリックします。

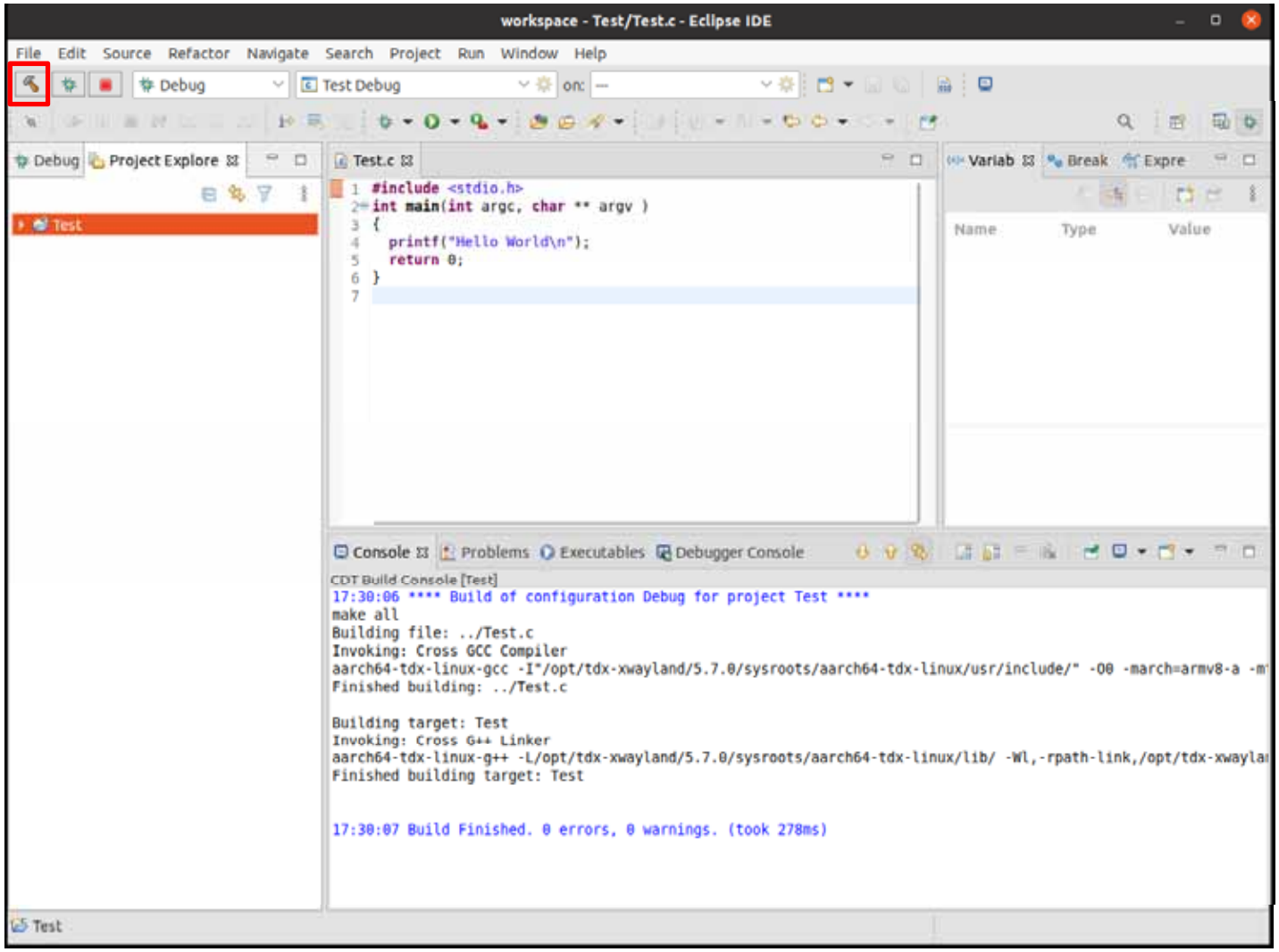


ソースコードを入力します。本マニュアルではHello Worldと出力するだけのソースコードを入力しています。

```
#include <stdio.h>
int main(int argc, char ** argv )
{
  printf("Hello World¥n");
  return 0;
}
```



ビルドボタンを押してビルドを行います。Consoleにログが出力されます。正常に終わった場合、青字でログが出力されます。



ビルドログは下記のようになります。
もしエラーが出た場合にはなにかしらの設定ミスがあります。差異に注意してください。

17:30:06 **** Build of configuration Debug for project Test ****

make all

Building file: ../Test.c

Invoking: Cross GCC Compiler

aarch64-tdx-linux-gcc -I"/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/usr/include/" -O0 -march=armv8-a -mtune=cortex-a53 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"Test.d" -MT"Test.o" -o "Test.o" "../Test.c"

Finished building: ../Test.c

Building target: Test

Invoking: Cross G++ Linker

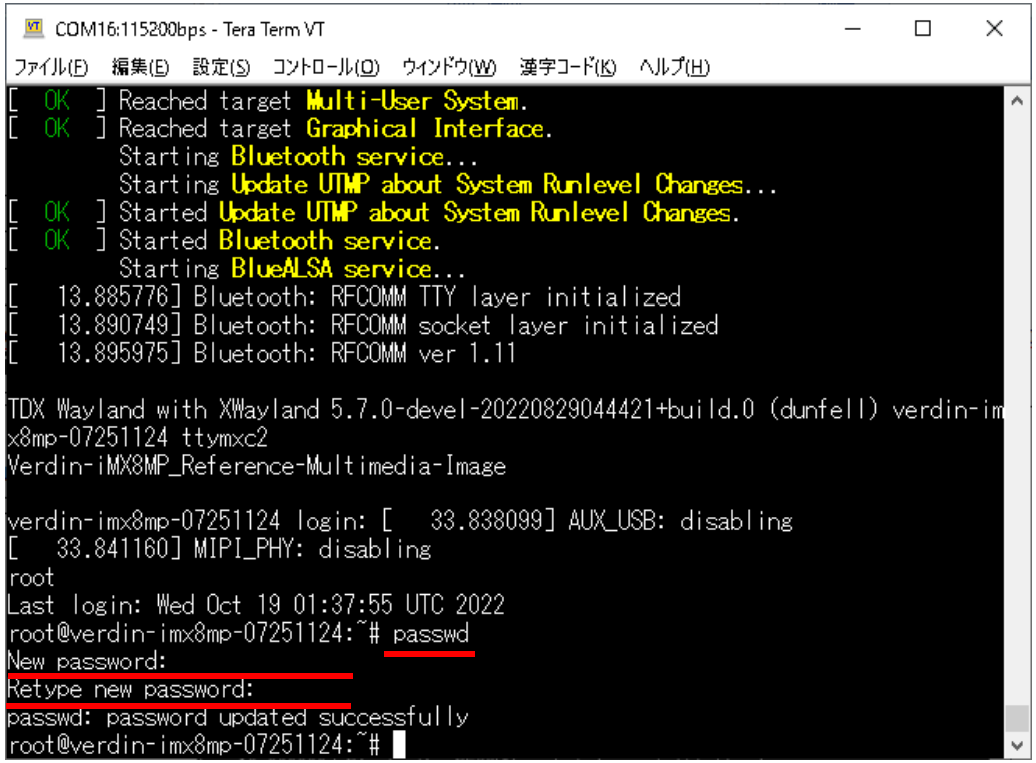
aarch64-tdx-linux-g++ -L/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/lib/ -Wl,-rpath-link,/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/lib/ -L/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/usr/lib/ -Wl,-rpath-link,/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/usr/lib/ --sysroot=/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux -o "Test" ./Test.o

Finished building target: Test

17:30:07 Build Finished. 0 errors, 0 warnings. (took 278ms)

SSH接続設定作成

デバッグに使用するGDBはEthernetで接続してSSHプロトコルを利用してデバッグを行います。
デバッグを行うためにモジュールとＳＳＨで接続できるようにしておく必要があります。
モジュール側の設定を行います。モジュールにEthernetケーブルを挿入し開発パソコンと同じサブネットに接続します。
デバッグコンソールを使ってコマンドを入力します。
Teraterm(Ubuntuの場合minicomなど)を起動してからモジュールを起動します。rootでログインしpasswdコマンドでパスワードを設定します。パスワードを2回入力するとパスワードが設定されます。(2回目は確認用)
本マニュアルではセキュリティを一切気にせず利便性のよいパスワード認証を使い、rootでSSHにログインできるようにします。
あくまでデバッグ目的で設定するだけです。
[Module]# passwd



次にモジュールのIPアドレスの設定を行います。何も設定していない場合はDHCPとなります。
設定ファイル/etc/systemd/network/wired.networkを新規作成します。

[Module]# vi /etc/systemd/network/wired.network

DHCPの場合

[Match]
Name=eth0

[Network]
DHCP=ipv4

---

eth0はインターフェイス名です。Verdin-iMX8M Plusはeth0とeth1があります。
モジュールやBSPのバージョンによって異なりますのでifconfigコマンドで調べてください。

固定IPの場合

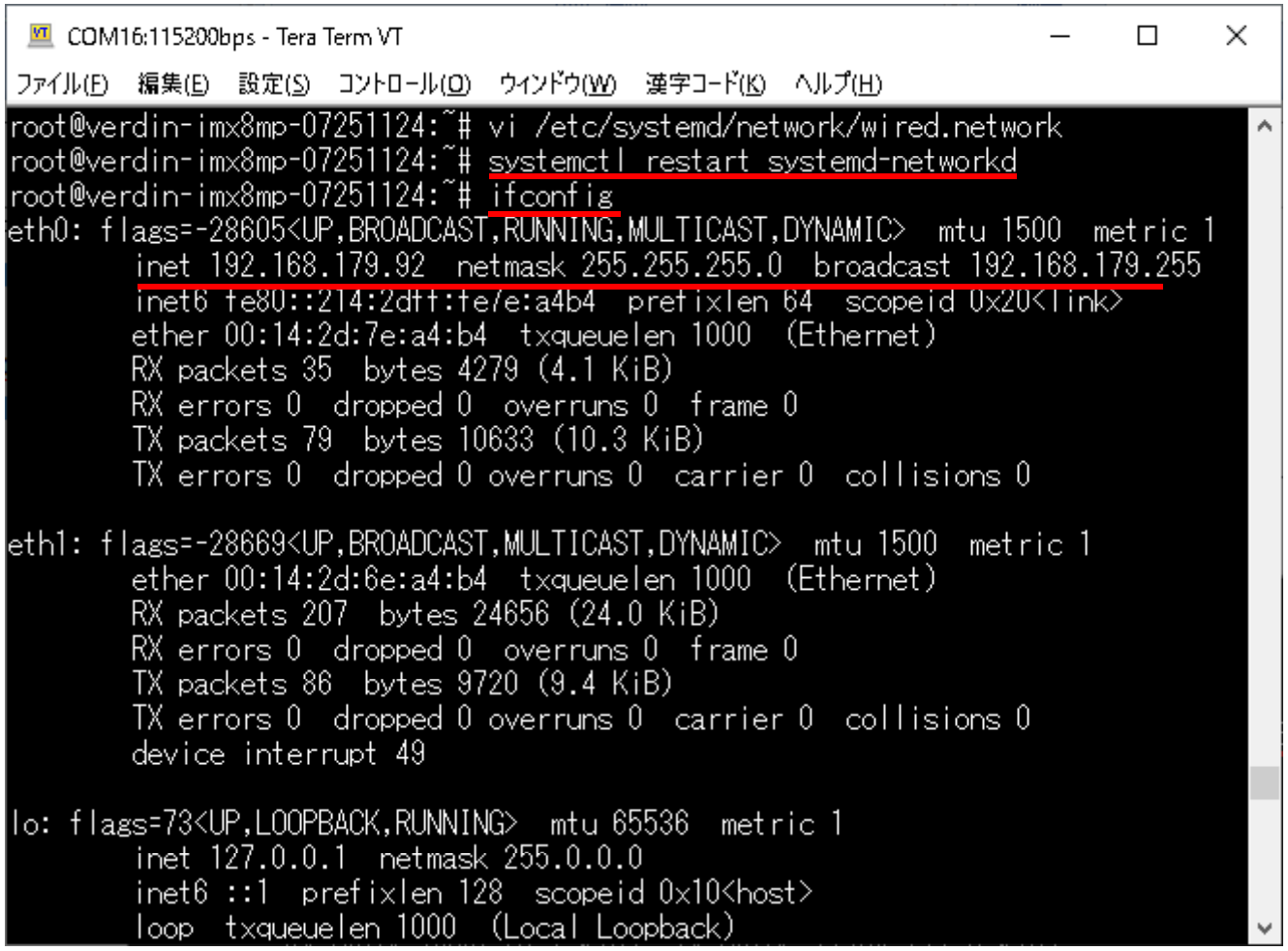
[Match]
Name=eth0

[Network]
Address=192.168.179.92/24
Gateway=192.168.179.1

192.168.179.92/24はIPアドレス192.168.179.92 サブネットマスク255.255.255.0を意味します。
IPアドレスは環境に応じて設定してください。

下記コマンドでネットワークマネージャーを再起動します。
[Module]# systemctl restart systemd-networkd

ifconfig**で設定が反映されているのを確かめます。**
[Module]# ifconfig

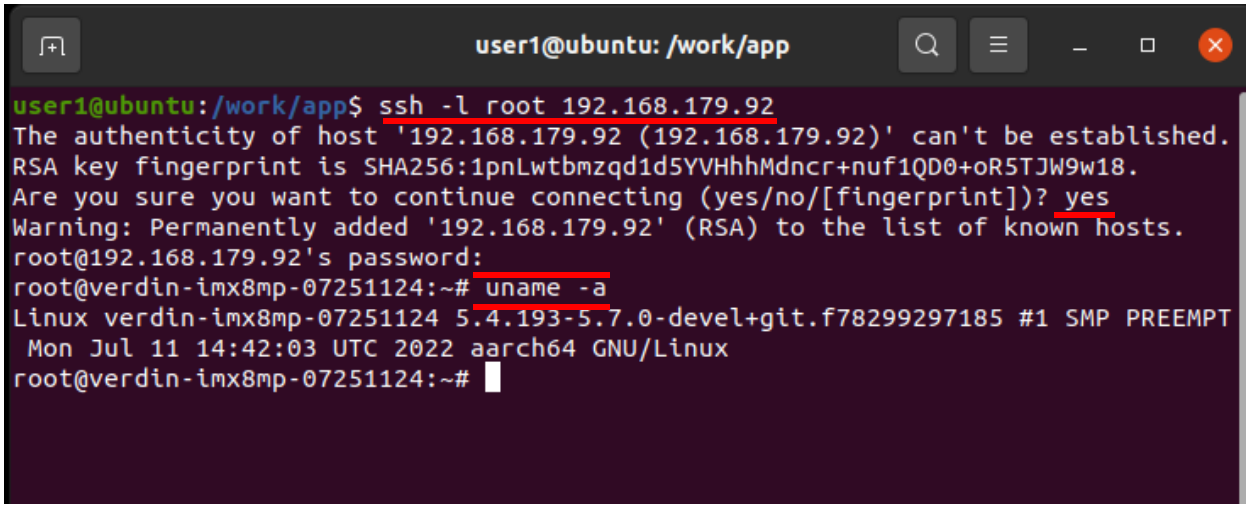


SSH接続確認

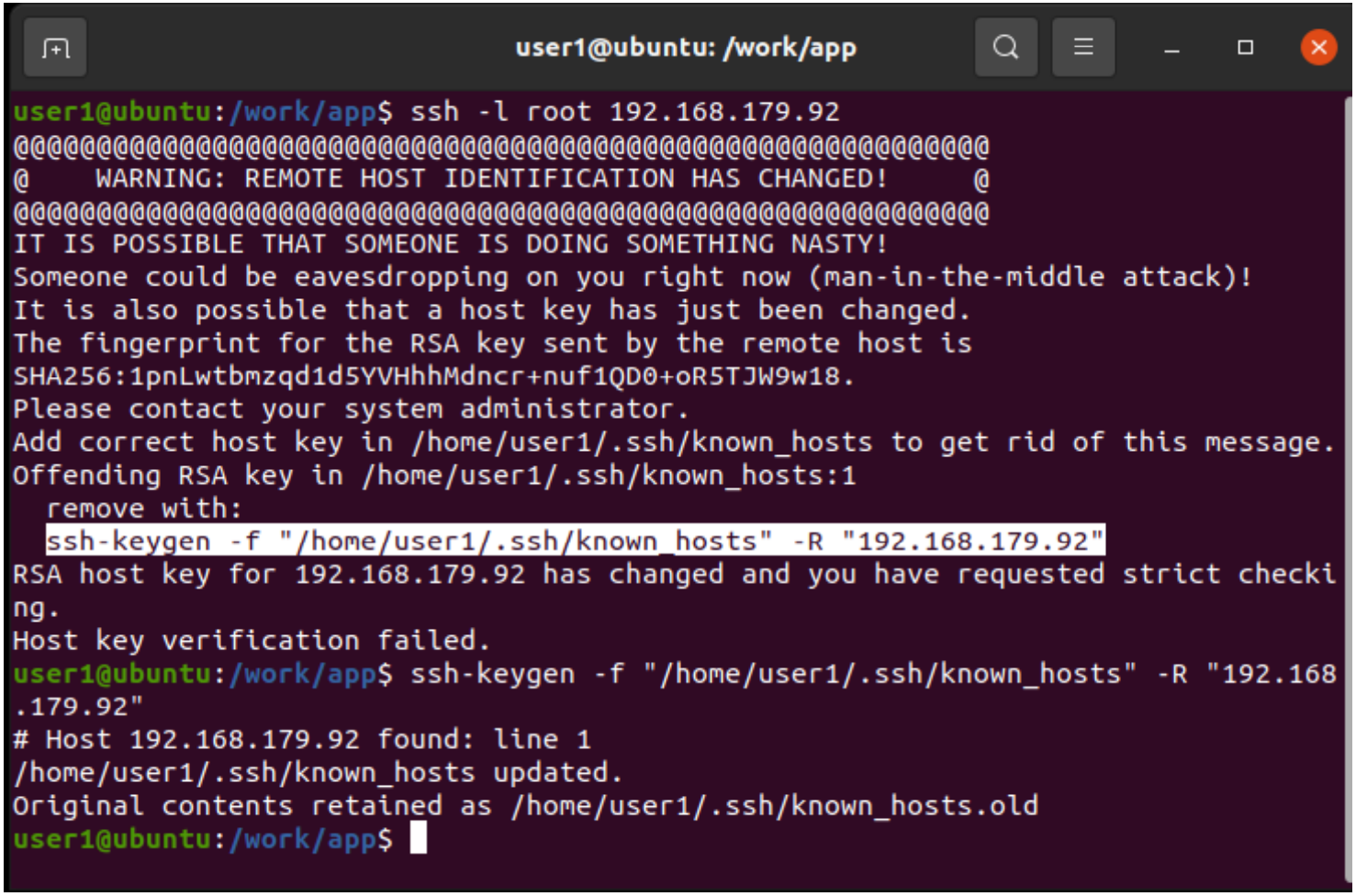
ホストOSやゲストからpingで開発パソコンから接続できているか確認します。
VMwareの設定がデフォルトのNATになっている場合、ホストOSのWindows10で接続できていればUbuntu側で接続できます。
接続できない場合は何かしらの設定が間違っている可能性があります。
[Ubuntu]$ ping XXX.XXX.XXX.XXX

sshコマンドで接続できるか試します。
[Ubuntu]$ ssh -l root XXX.XXX.XXX.XXX

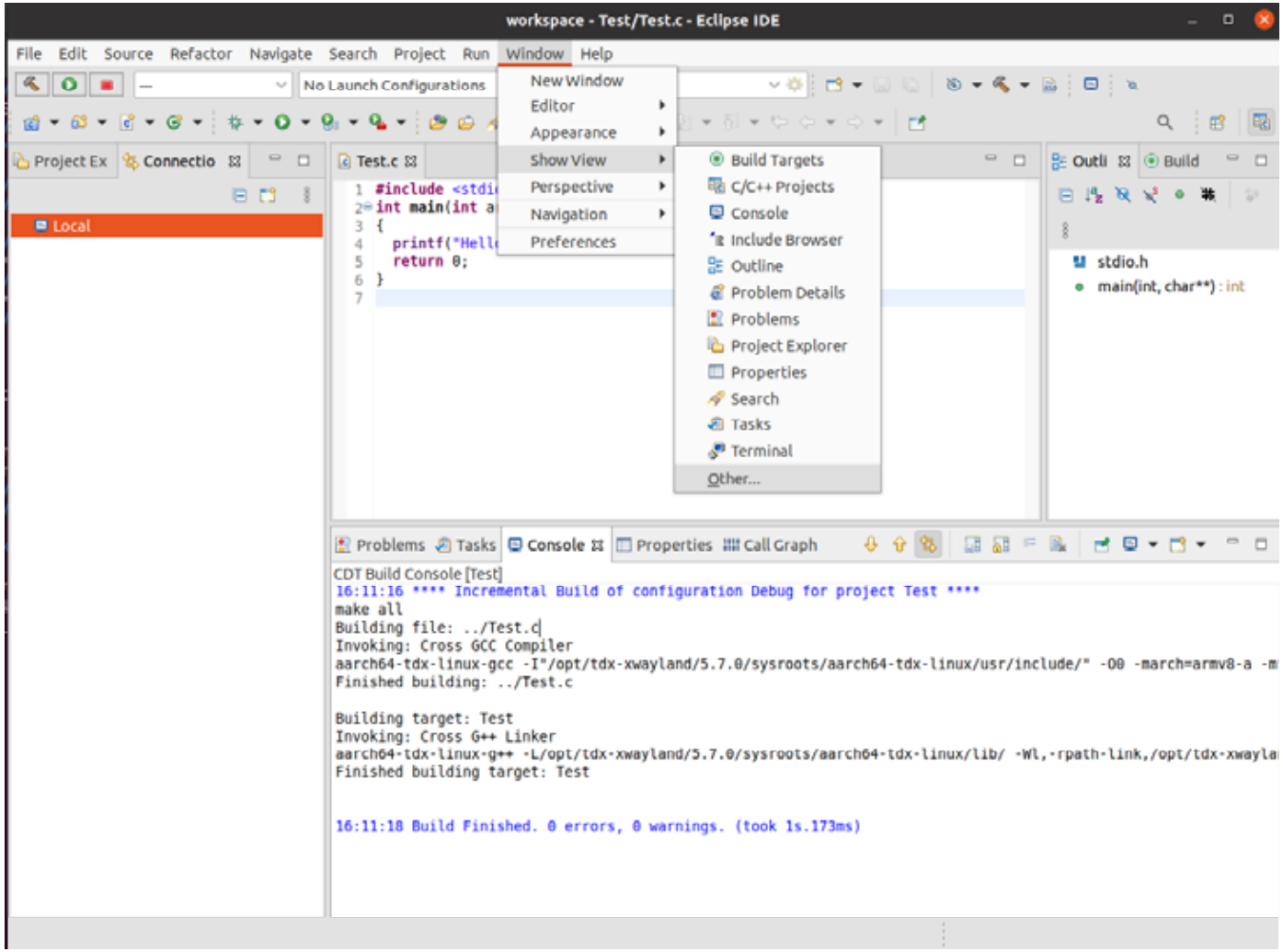
最後にunameコマンドでログインできているかを確かめています。
[Module]# uname -a



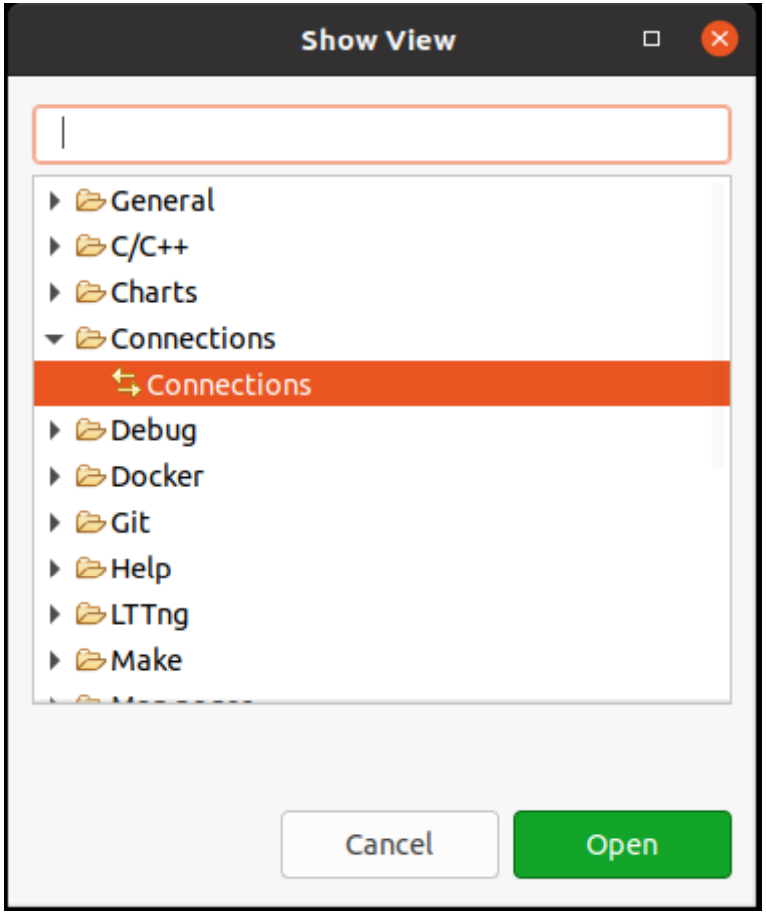
sshコマンドで下記のようなエラーが出た場合
ssh -keygenでキーを一度削除してください。



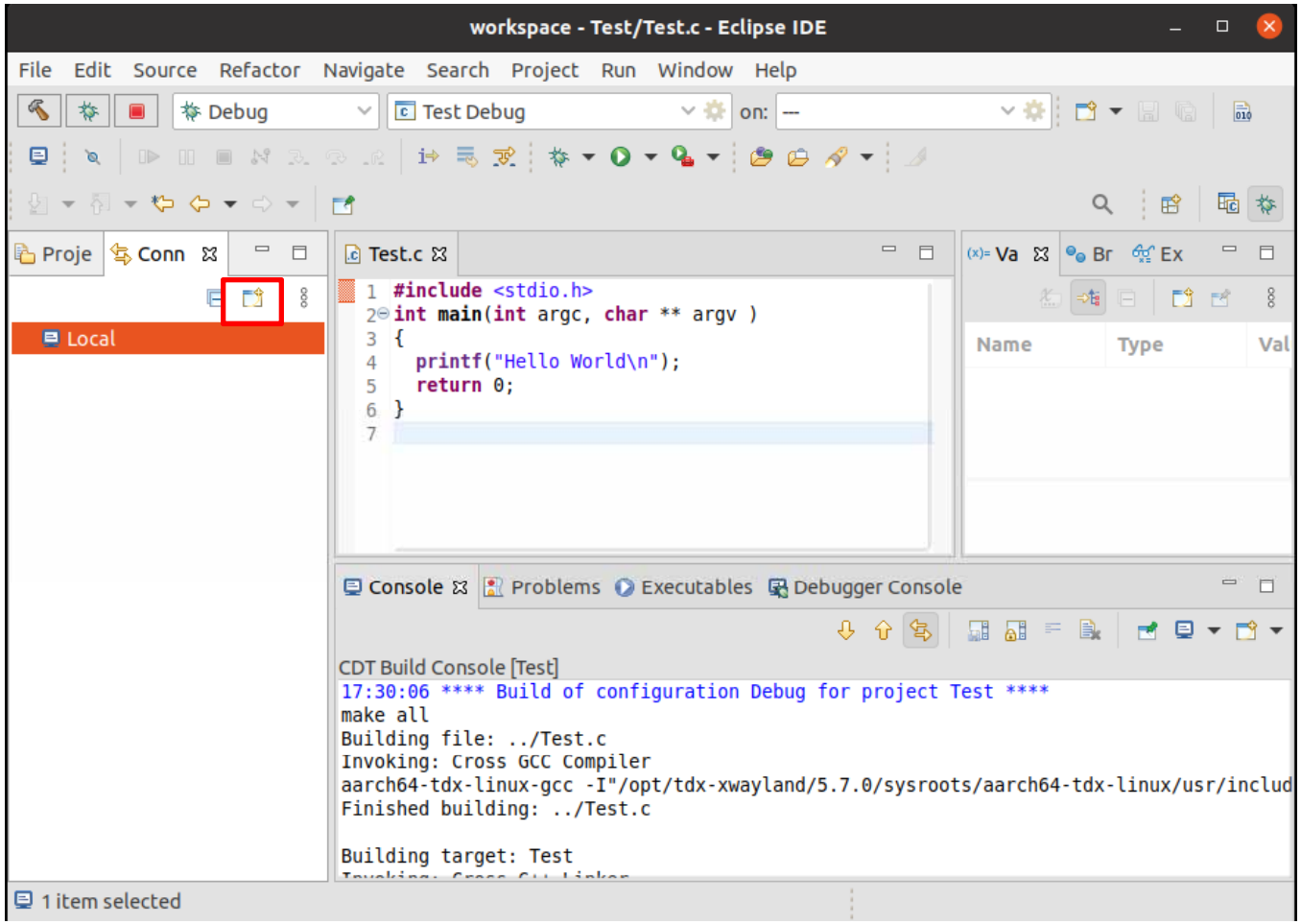
Eclipseに戻ります。Eclipseのメニューの Window > Show View > Otherを選択します。



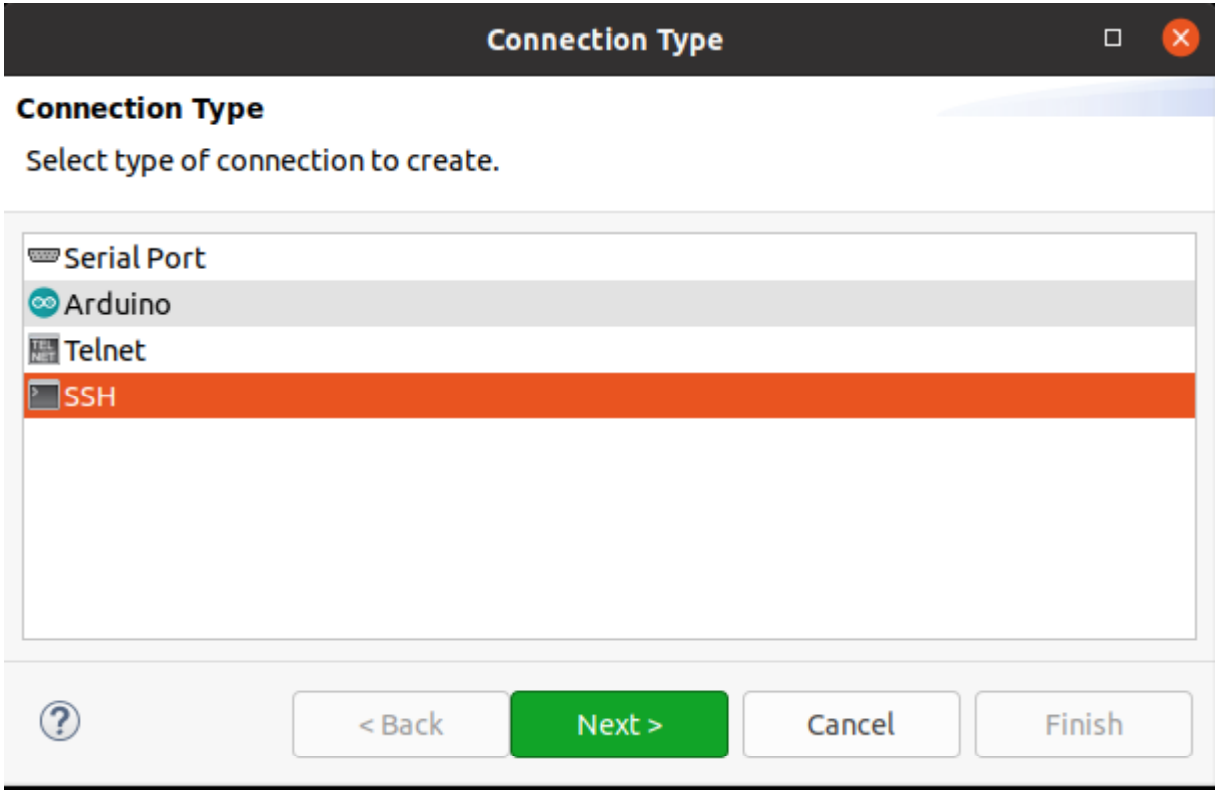
Connectionsを選択してOpenをクリックします。



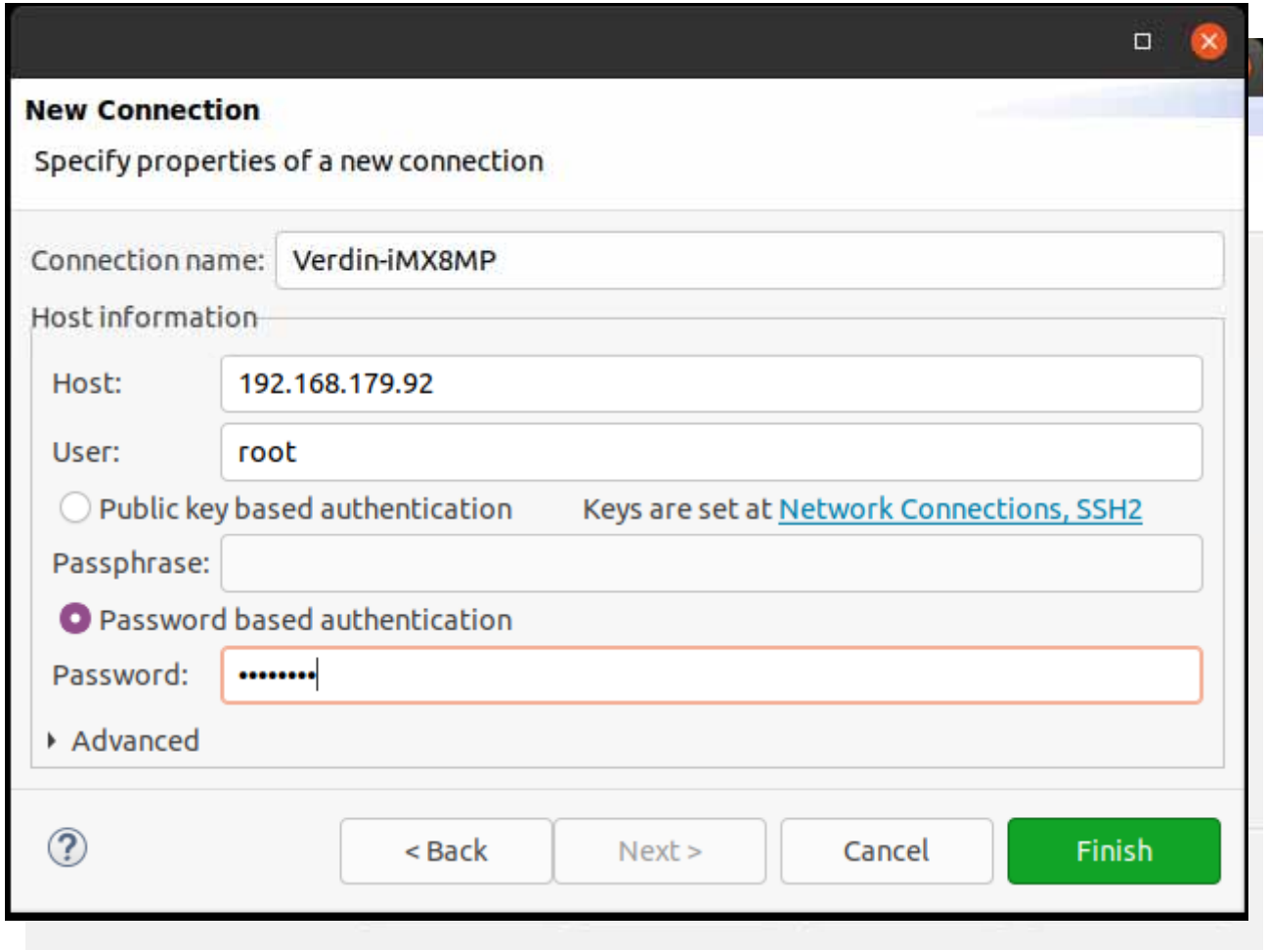
＋マーク(赤枠)のアイコンをクリックしてConnection設定を追加します。



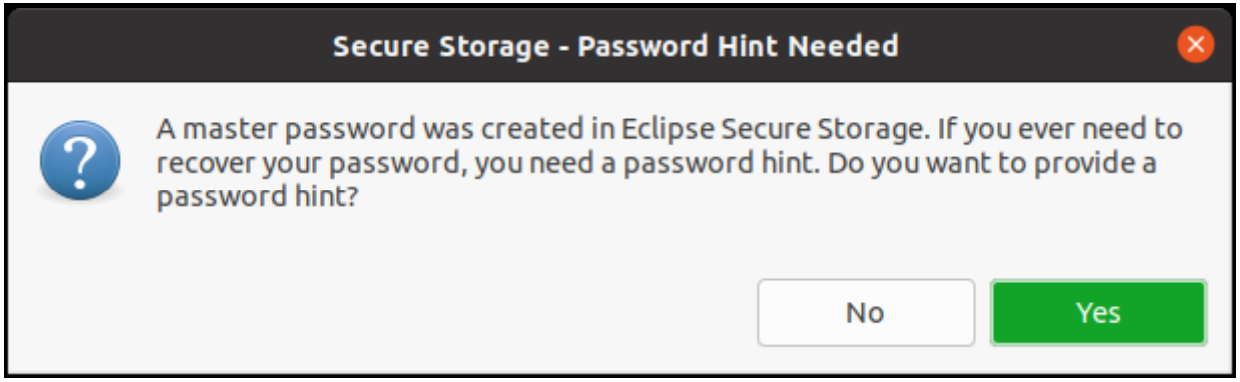
SSHを選択してNextをクリックします。



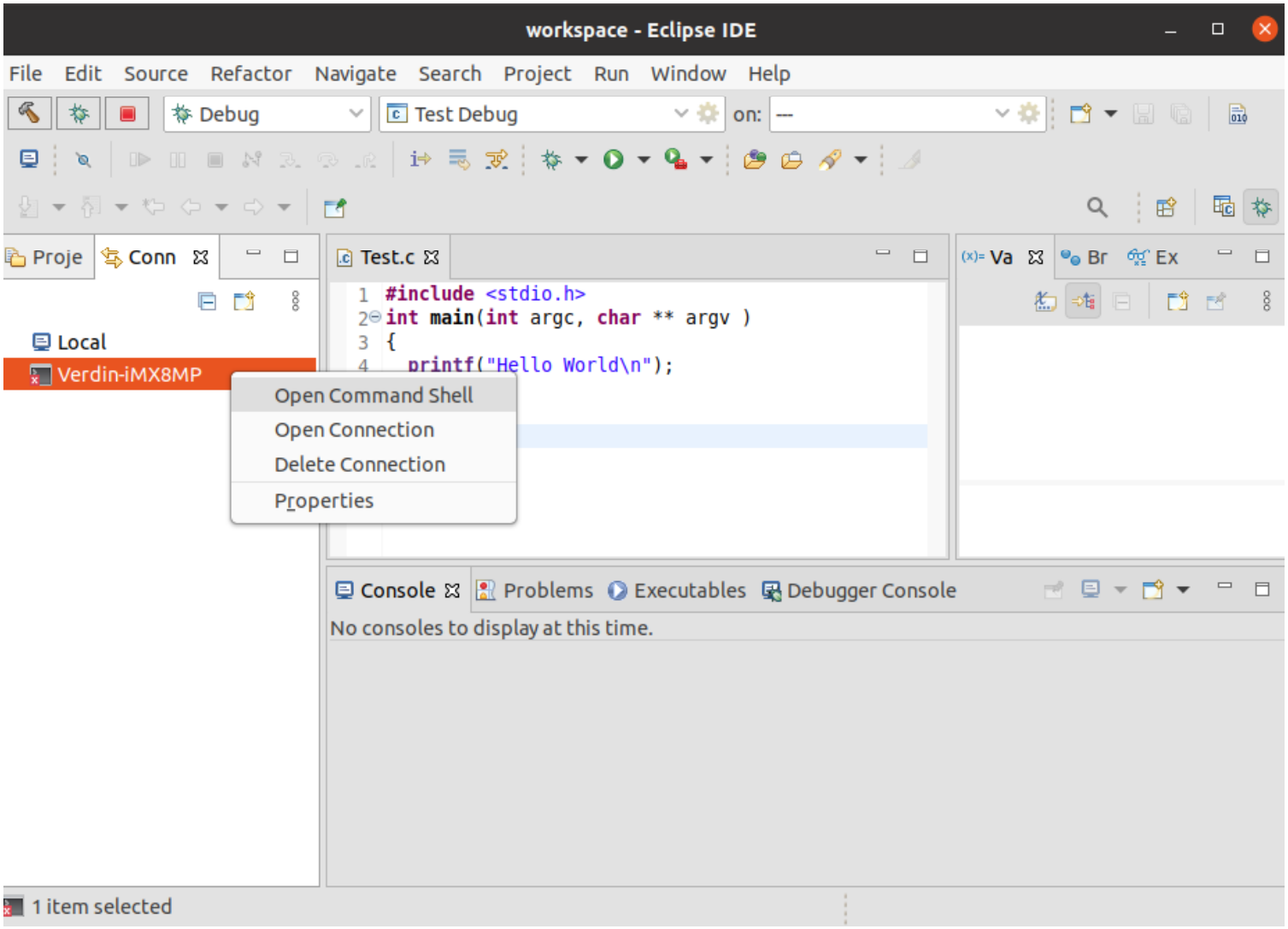
Connection nameにわかりやすい名前を付けます。
Host, User,Passwordにそれぞれモジュールのipアドレス、ユーザー名、パスワードを入力してFinishをクリックします。



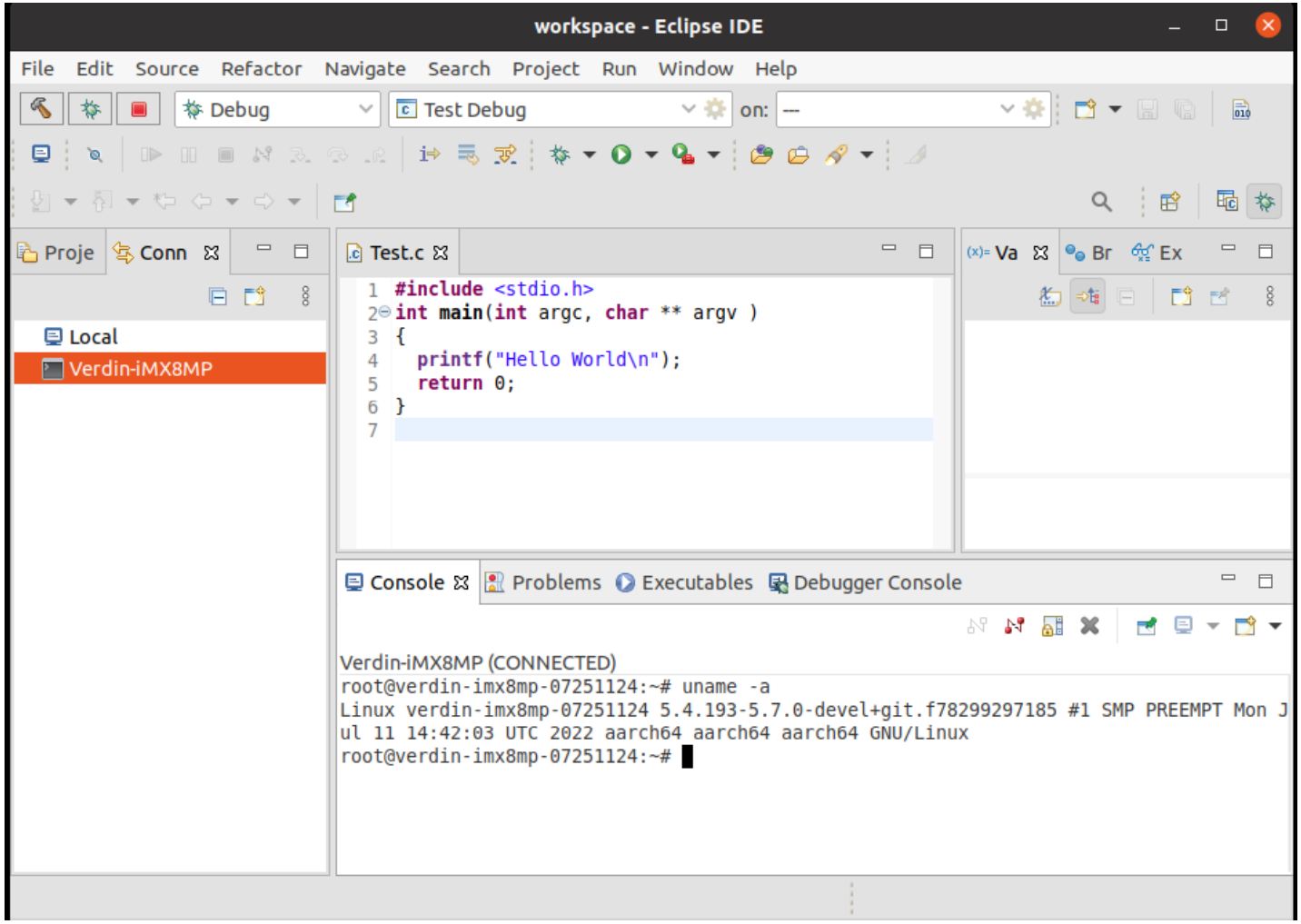
パスワードの保存を行った場合、パスワード復元用のヒントを作るかどうかを問われます。本マニュアルでは使用しないのでNo
をクリックします。



作成したConnection設定を右クリックしてOpen Command Shellを選択します。



モジュールと接続できるとConsoleでコマンドが打てるようになります。下記ではunameコマンドを実行しています。

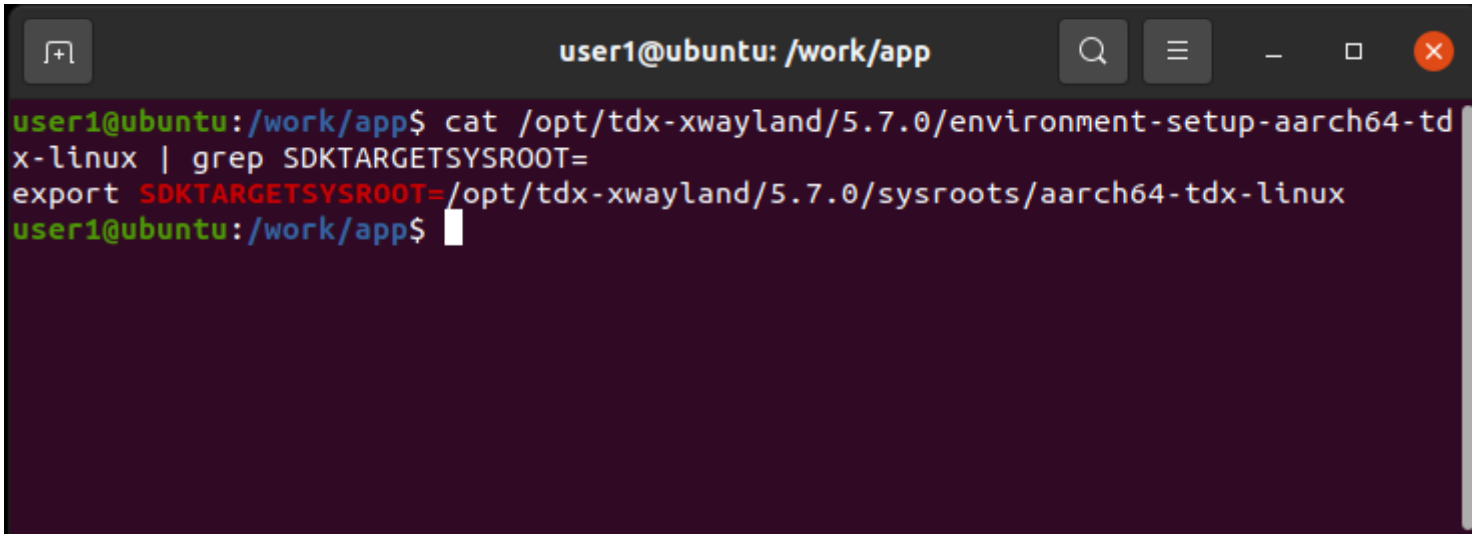


デバッグ設定作成

GDBの初期処理を記述するgdbinitファイルを作成します。本マニュアルでは/work/app/gdbinit　に作成します。
[Ubuntu]$ gedit /work/app/gdbinit
内容は下記です。(環境変数は使用できません。)
set sysroot /opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux
set auto-load safe-path /opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux

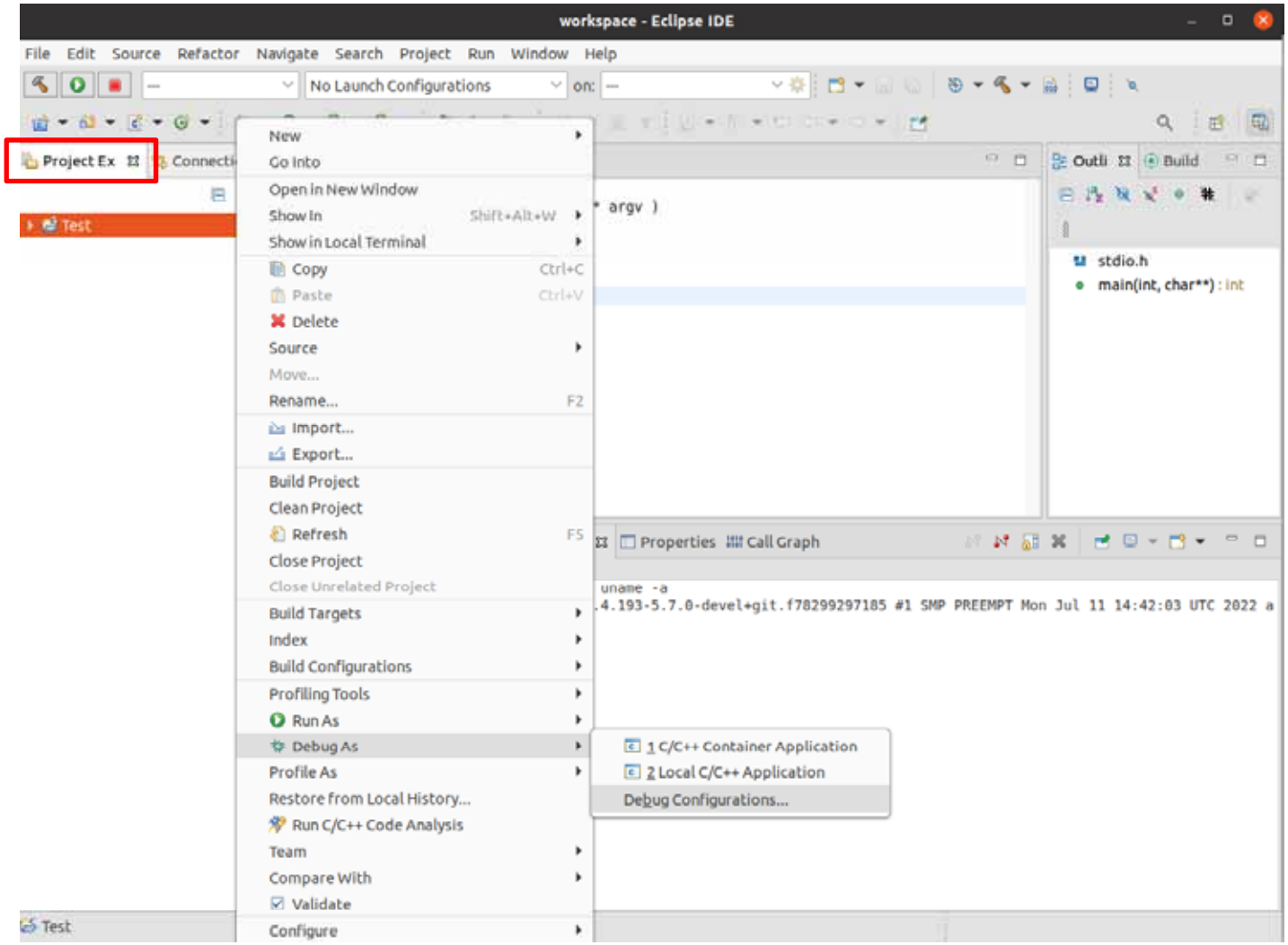
上記2つのコマンドで指定するパスはSDKが出力した環境変数設定シェル内の下記の内容になります。
SDKTARGETSYSROOT

下記コマンドで見ることができます。
[Ubuntu]$ cat /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux | grep SDKTARGETSYSROOT=

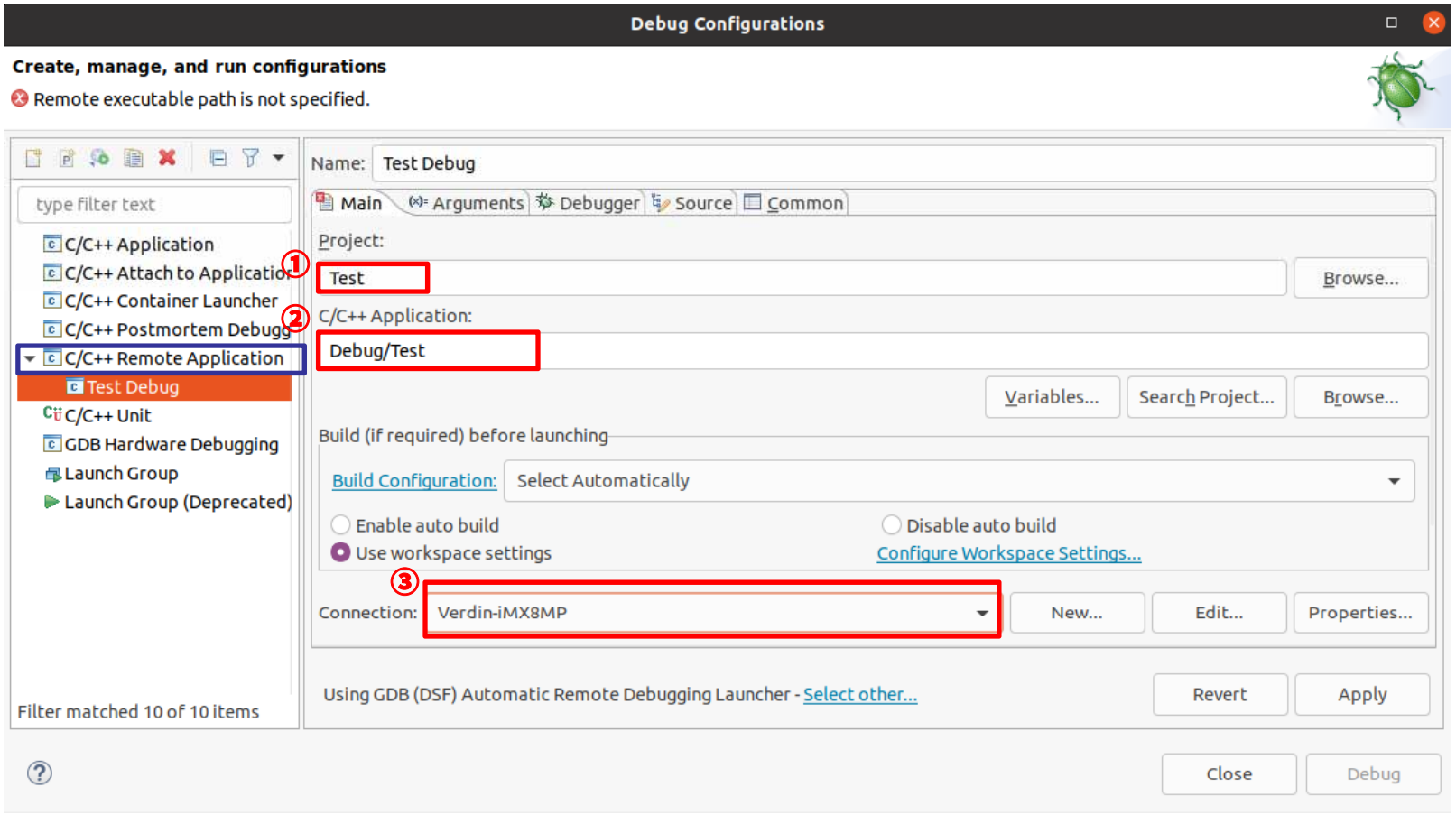


デバッグ

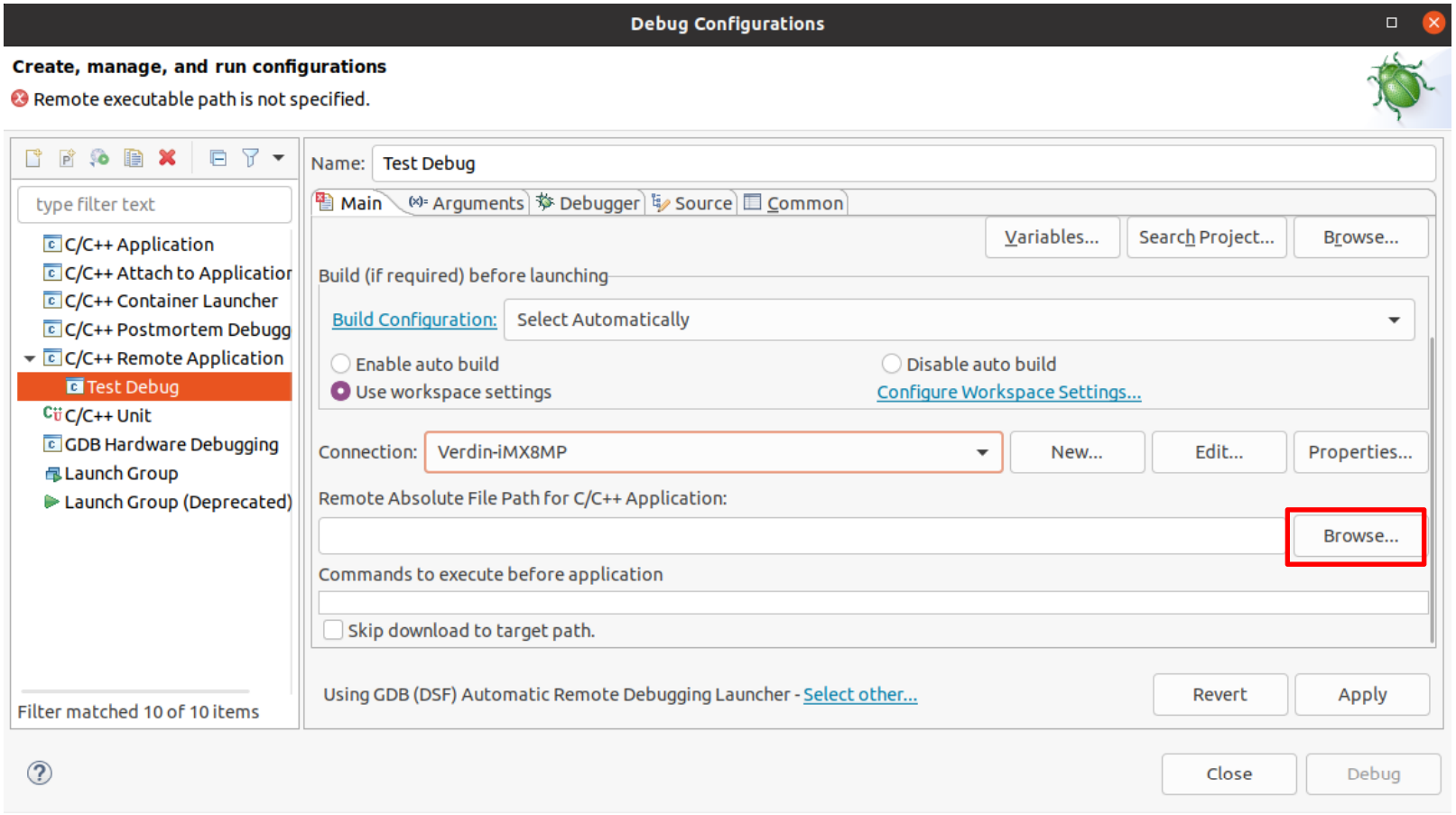
プロジェクトエクスプローラに戻り(赤枠をクリック)、プロジェクトを右クリックしDebug As > Debug Configurationsを選択します。



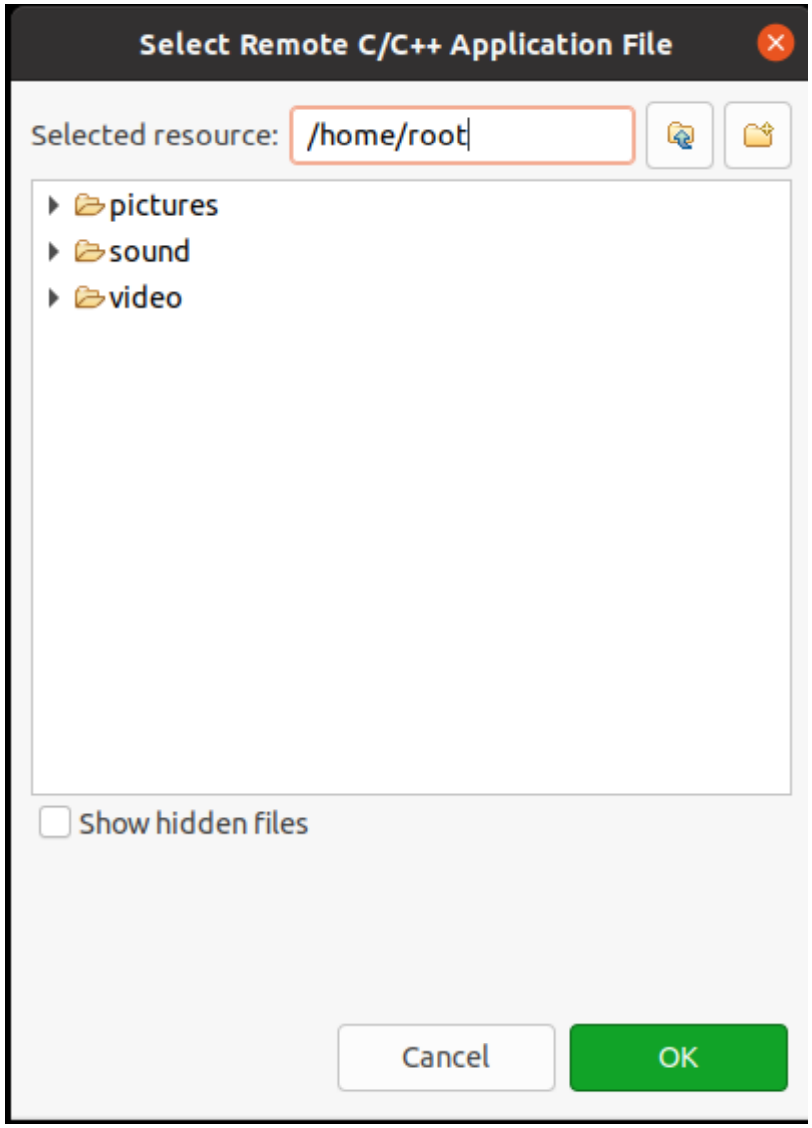
C/C++ Remote Application(青枠)をダブルクリックしデバッグ設定を追加します。赤枠内の設定を行います。
①Projectと②C/C++ Applicationは自動で設定されるままで問題ありません。
③Connectionは先ほど作成した設定を選択します。



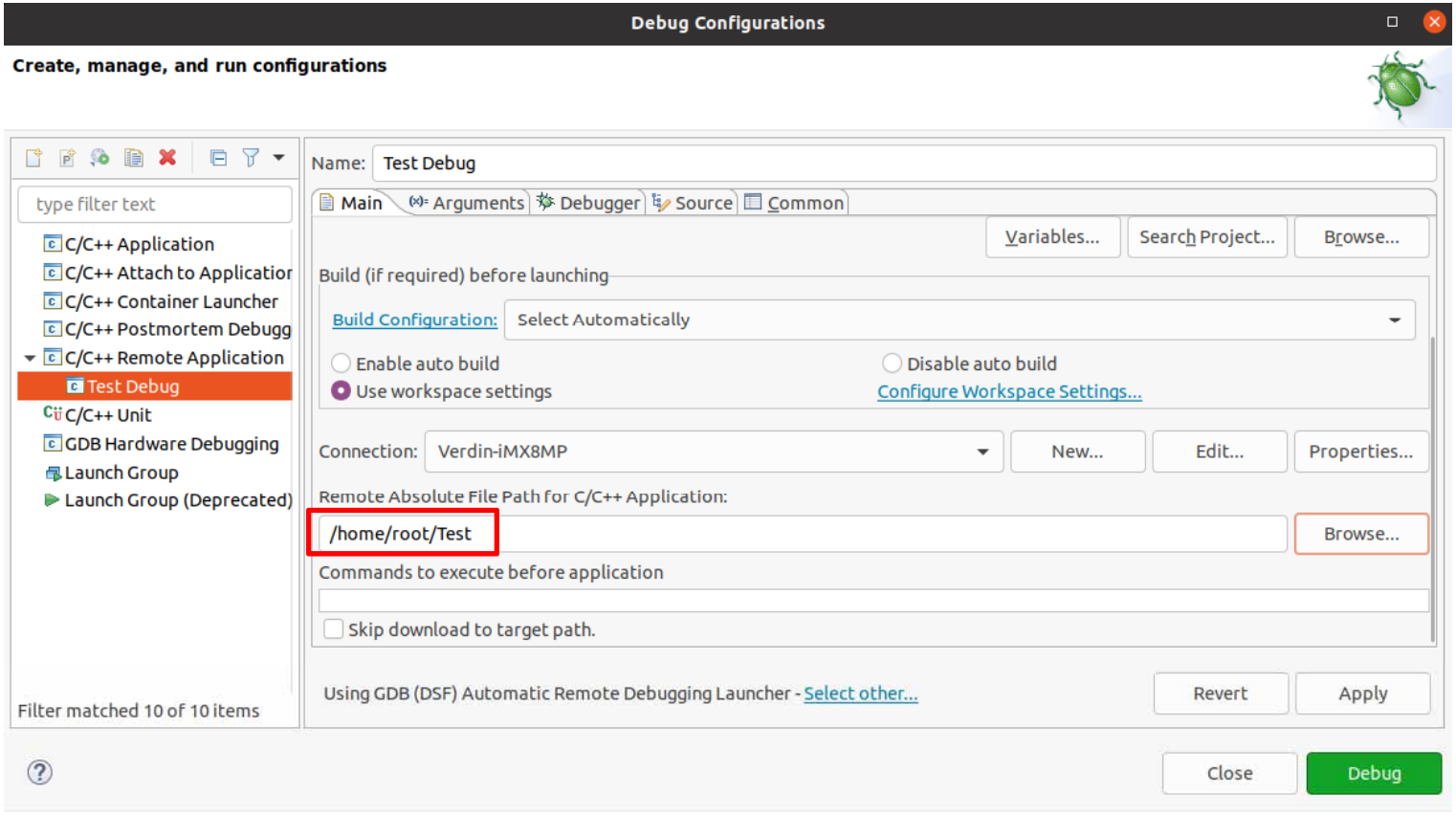
Browseをクリックします。



接続ができていればモジュール内のホームディレクトリが見えます。そのままOKをクリックします。



自動的にRemote Absolute File Path for C/C++ Application**に実行パスが設定されます。**

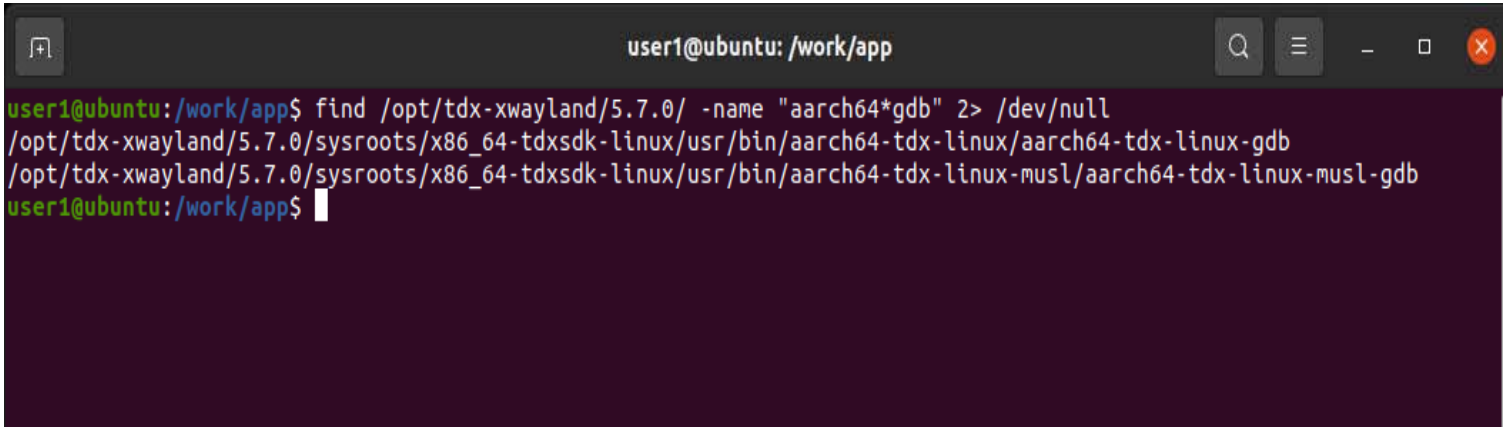


gdbのパスを確認します。

上記で指定するパスはSDK配下をaarch64用のgdbを探します。
find /opt/tdx-xwayland/5.7.0/ -name "aarch64*gdb" 2> /dev/null

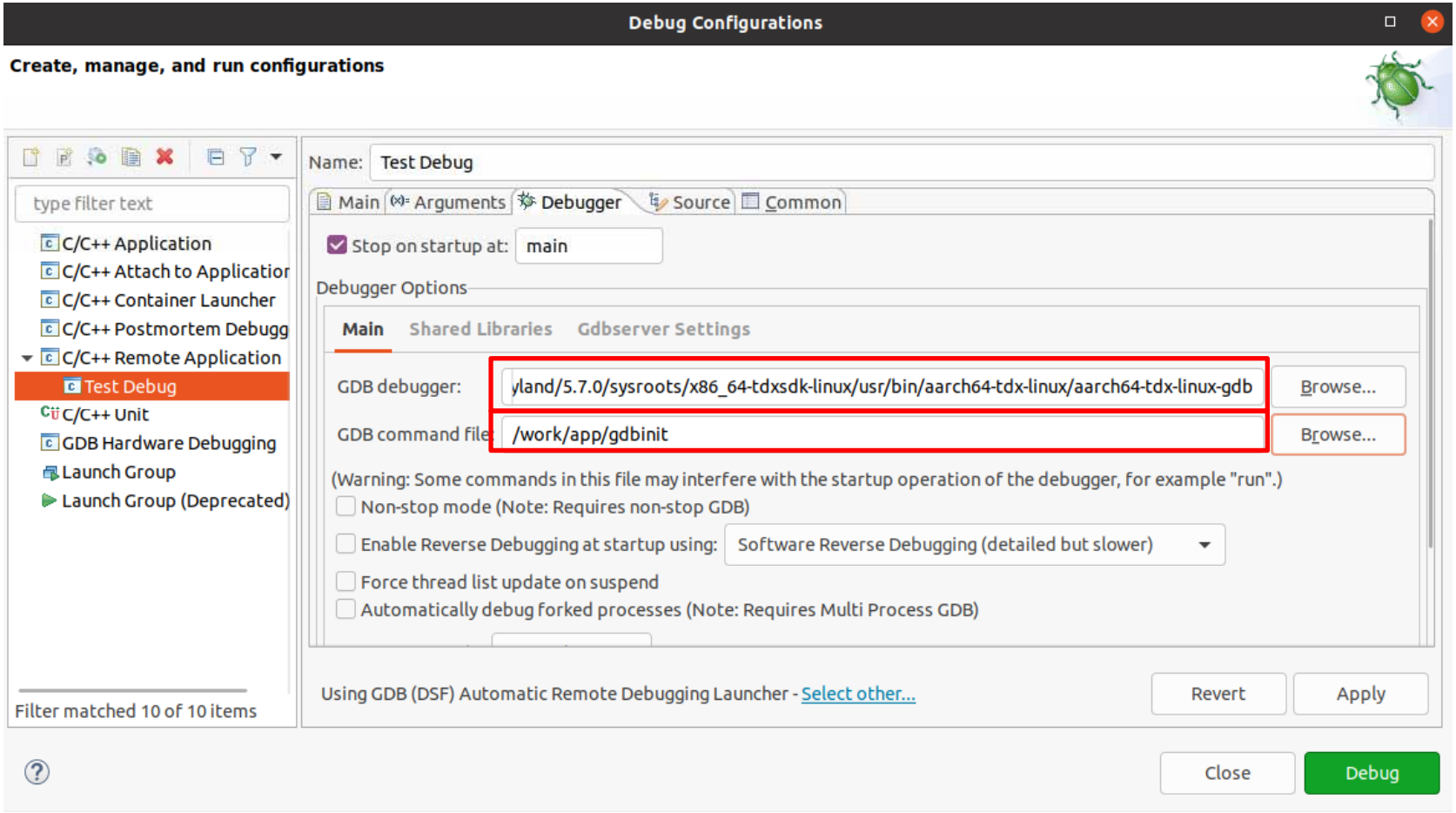
使用するのはmuslがついていないほうです。

本マニュアルでは下記になります。
/opt/tdx-xwayland/5.7.0/sysroots/x86_64-tdxsdk-linux/usr/bin/aarch64-tdx-linux/aarch64-tdx-linux-gdb

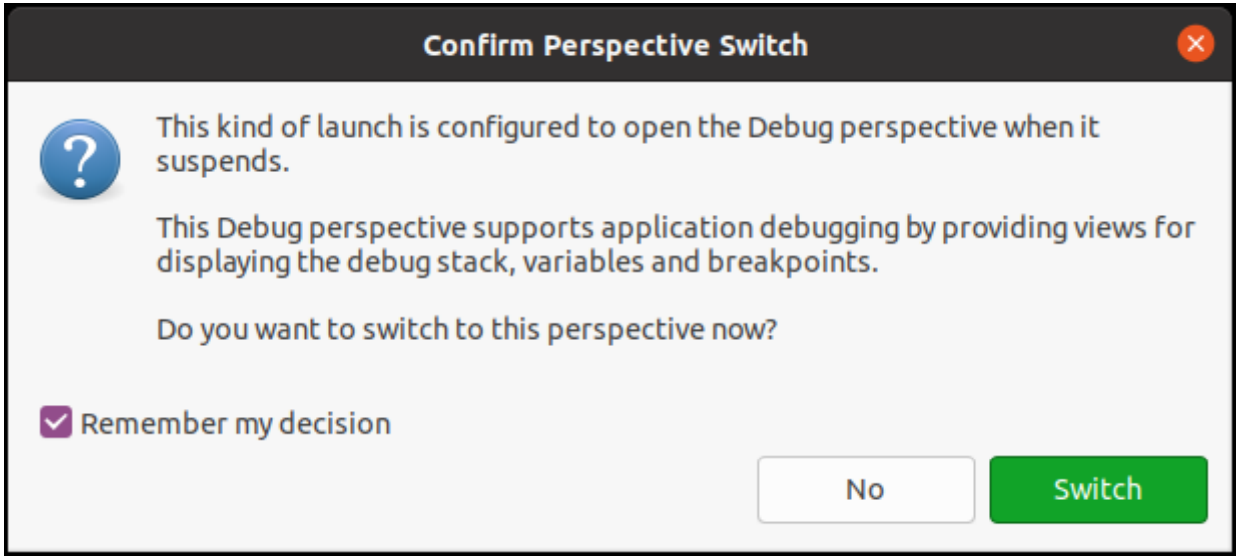


Debuggerのタブを開きます。GBD debugerに下記を指定します。
/opt/tdx-xwayland/5.7.0/sysroots/x86_64-tdxsdk-linux/usr/bin/aarch64-tdx-linux/aarch64-tdx-linux-gdb

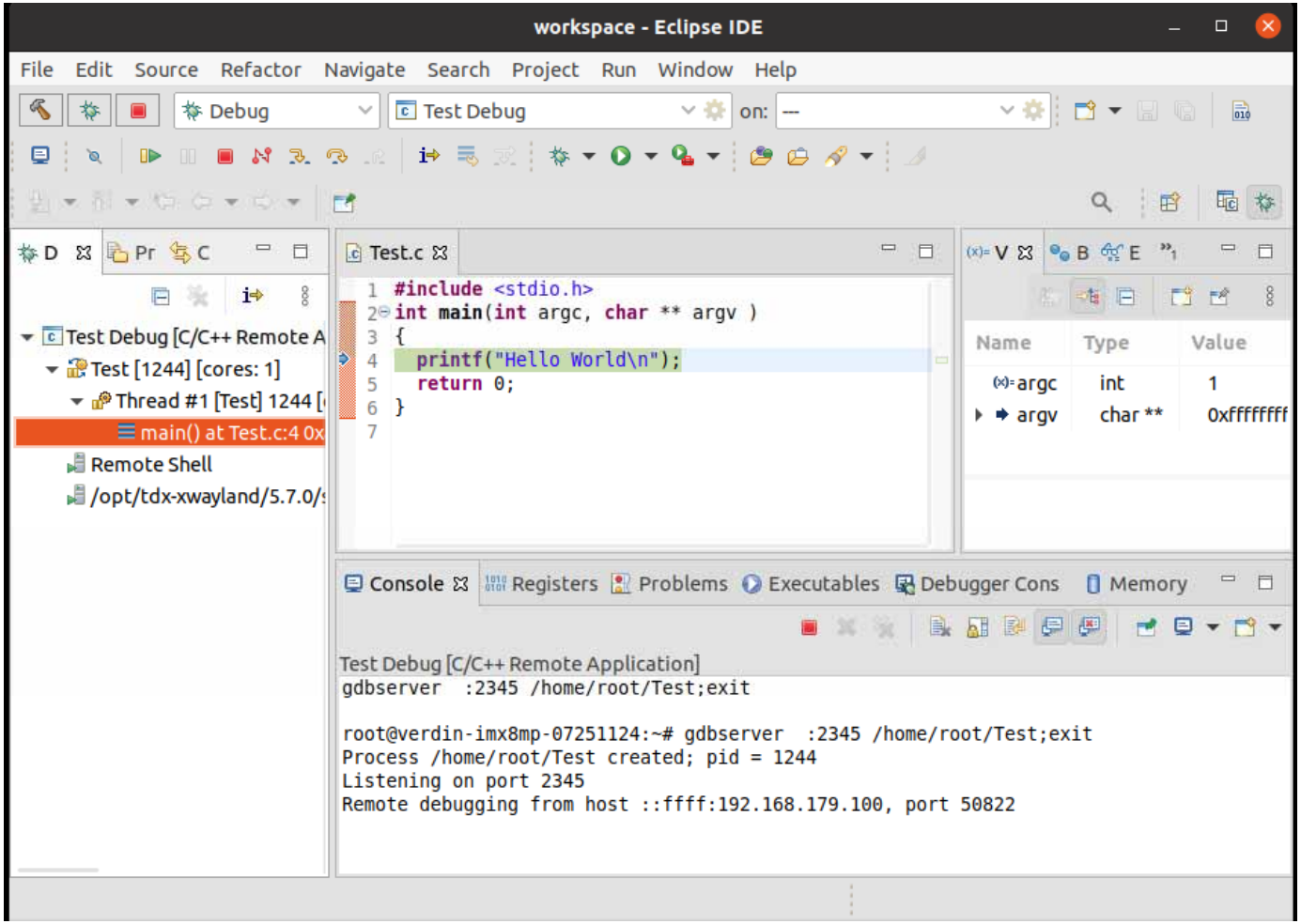
GDB command fileには先ほど作成したgdbinitのパスを指定します。本マニュアルでは下記になります。
/work/app/gdbinit
Applyをクリック後Debugをクリックします。



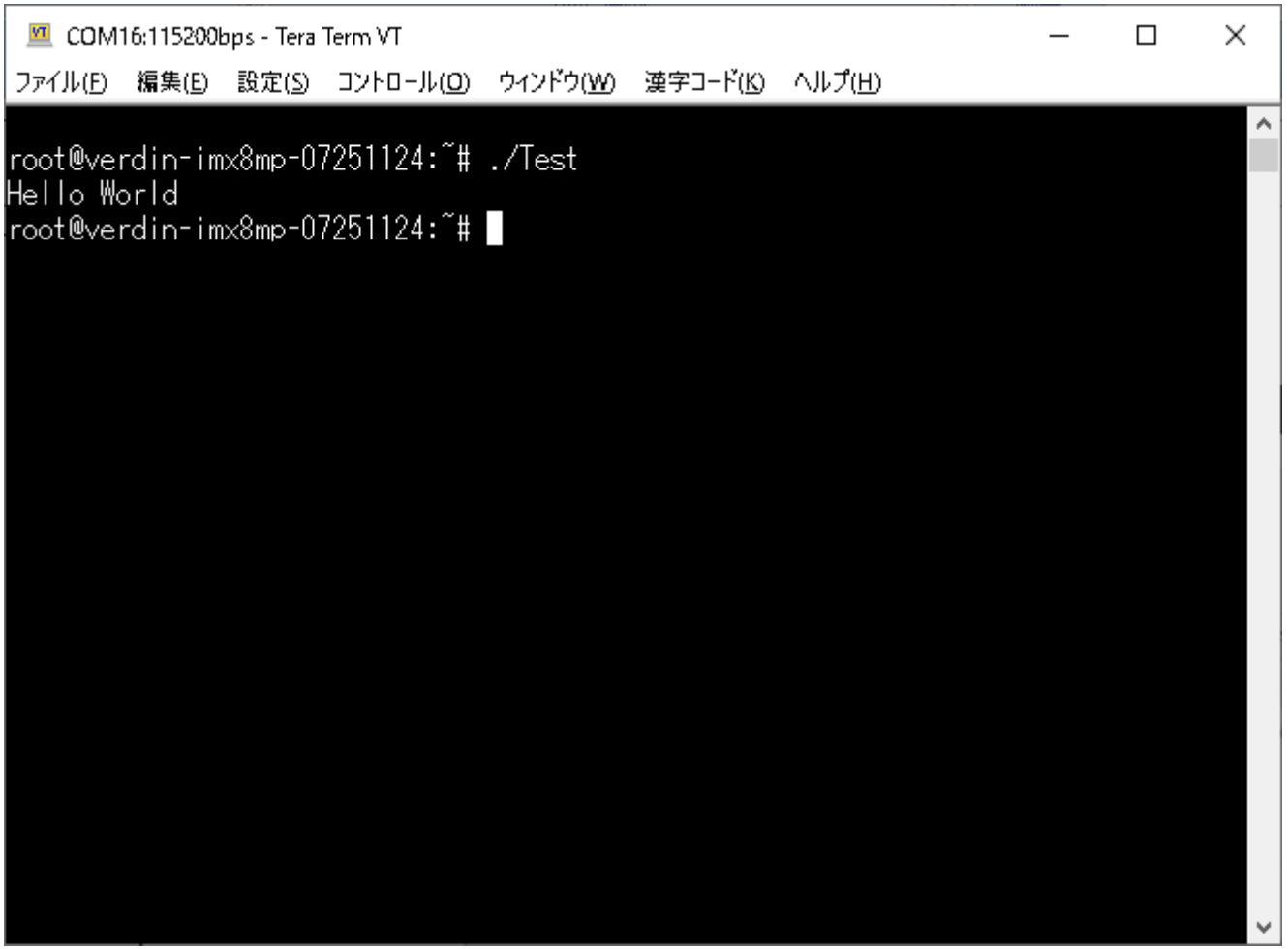
Debugが始まると下記のようなDebugウィンドウを開くかどうか問われますのでSwitchをクリックします。
この操作を覚える場合はRemember my decisionにチェックを入れておきます。



デバッグウィンドウが表示されデバッグを行うことができます。



デバッグを行うと自動的に実行環境の/root/homeに実行ファイルがコピーされています。実行可能です。



Releaseビルドで実行ファイルを作成

デバッグを行いプログラムを完成させた後はリリースビルドで最適化を行ったバージョンの実行ファイルを作成します。
プロジェクトを右クリック Build Configurations > Set Active > Releaseを選択してReleaseに変更後ビルドボタンを押しビルドを行います。ビルドした実行ファイルは/work/app/workspace/Test/Releaseに格納されています。
この実行ファイルをSDカードなどでモジュールに移動し実行することができます。

以上がC/C++言語アプリケーションの開発の環境構築からデバッグ、実行ファイル作成までの手順です。