Visual Studio:

Attach to a running process

1. On the Debug menu, select Attach to Process.

2. In the Attach to Process dialog box, find the program that you want to attach to from the Available Processes list.
   a. If the program that you want to debug is running on another computer, use the Qualifier list box to select or specify the remote computer. For more information, see Remote Debugging and Diagnostics.
   b. If the process is running under a different user account, select the Show processes from all users checkbox.
   c. If you are connected through Remote Desktop Connection, select the Show processes in all sessions check box.
3. In the Attach to box, make sure that the type of code you will debug is listed. The default Automatic setting tries to determine what type of code you want to debug. If the automatic setting is not appropriate:
   a. Click Select.
   b. In the Select Code Type dialog box, click Debug these code types and select the types to debug.
   c. Click OK.
4. Click Attach.
5. The **Available Processes** list is displayed automatically when you open the **Processes** dialog box. Processes can start and stop in the background while the dialog box is open. However, the contents are not always current. You can refresh the list at any time to see the current list of processes by clicking **Refresh**.

6. You can be attached to multiple programs when you are debugging, but only one program is active in the debugger at any time. You can set the active program in the **Debug Location** toolbar or the **Processes** window.

### Attach to a process on a remote computer

When you use the **Attach to Process** dialog box, you can select another computer that has been set up for remote debugging. For more information, see Set Up the Remote Tools on the Device. When you have selected a remote computer, you can view a list of available processes running on that computer and attach to one or more of the processes for debugging.

**Note:** To set up the Remote Debugging follow these steps:

https://msdn.microsoft.com/en-us/library/bt727f1t(v=vs.120).aspx

You will also need to download and install this:

To select a remote computer:

1. In the Attach to Process dialog box, select the appropriate connection type from the Transport list. Default is the correct setting for most cases.
2. Use the Qualifier list box to choose the remote computer name by one of the following methods:
   a. Type the name in the Qualifier list box.
   b. Click the drop-down arrow attached to the Qualifier list box and select the computer name from the drop-down list.
   c. Click the Find button next to the Qualifier list to open the Select Remote Debugger Connection dialog box. The Select Remote Debugger Connection dialog box lists all the devices that are on your local sub-net, and any device that is directly attached to your computer through an Ethernet cable. Click the computer or device that you want, and then click Select.
3. The Transport setting persists between debugging sessions. The Qualifier setting persists between debugging sessions only if a successful debugging connection occurs with that qualifier.

All steps taken from https://msdn.microsoft.com/en-us/library/3s68z0b3.aspx
Eclipse:
The following is setup with Eclipse and Java, although there are similar ways to remote debug with C++ and other languages, although many will be harder and more specific to set up, as they are not languages Eclipse is best worked in.

From what I researched, I could not find a way that Eclipse allows to attach to a running process without that process broadcasting a debug server.

To run a java program while having it broadcast a debug server, add the following flags when starting the server (within command line):

```
-Xdebug -Xrunjdwp:transport=dt_socket,address=address:port,server=y,suspend=n
```

(These flags are described in more detail below under “Remote Debugging in NetBeans”.)

The address can be the address of another computer, or the localhost, the former being remote debugging across platforms, and the latter being local debugging.

One can also run the java program within Eclipse and have it broadcast a debug server through this IDE instead of through the command line.

Eclipse can then connect to and remotely debug on this server, so that one can add breakpoints and follow the code in Eclipse, while the code is executing outside of the IDE.

Remote Debugging in Eclipse

The workflow is as follows:

1. **Run** the **Java** program with the flags shown above, with address defining the desired address through the desired port.
   Example:
   ```
   java -jar -Xdebug
   -Xrunjdwp:transport=dt_socket,address=127.0.0.1:8888,server=y,suspend=n
   anagrams.jar
   ```

2. **Start Eclipse** and open (or create a duplicate of) the Java program code.
   a. Place whatever breakpoints throughout the code that you want to debug for.
3. On the Eclipse toolbar, go to **Run → Debug Configurations**, as shown here:

4. Right click on **Remote Java Application** and click **New**.
5. Change the **Name** to your desired name. Be sure the **Project** field is set to the Eclipse project that will be debugging. Be sure the **Connection Type** field is set to **Standard (Socket Attach)**. Fill in the same **Host** and **Port** as the server started above. This step, as an example, is shown below:

6. Click **Debug**, and you should now be remotely debugging.

   This workflow was to show that Eclipse is debugging remotely, separate from the running program. You can, however, run the program within Eclipse as the way to broadcast the server.
This workflow would then skip Step 1, and alter Step 5. In Step 5, Change the **Connection Type** field to **Standard (Socket Listen)**. The **Host** field will disappear, as the listening system is the host. You can then start up a separate instance of Eclipse, like on a separate system, to then attach to this listener and remote debug from there.

Source:
Netbeans:
Before doing any attaching to processes, Netbeans works best when the version of java being
run on this system is calibrated.

Instrumenting the bytecode of the profiled application imposes some overhead. To guarantee
the high accuracy of profiling results, NetBeans Profiler needs to collect calibration data in order
to "factor out" the time spent in code instrumentation. You need to run the calibration process for
each JDK you will use for profiling. The calibration data for each JDK is saved in the .nbprofile
directory in your home directory.

You are prompted to run the calibration the first time you invoke NetBeans Profiler. You are also
prompted if the calibration data for the local machine and JVM is unavailable.

Running the Calibration

When performing calibration, make sure that no other programs are running on the same
machine (or at least that no other program is performing any significant computation at the
time). Profiler will run the calibration regardless, but running any CPU-intensive programs when
performing the calibration may affect the accuracy of profiling results.

To calibrate, follow these steps::

- **Calibration for local profiling.** To collect calibration data for profiling on your local
  system, choose Profile > Advanced Commands > Run Profiler Calibration from the main
  menu. The following dialog is displayed:

  ![Manage Calibration Data](image)

  - Choose the Java Platform that will be used for profiling and click OK. A dialog box
    appears when the calibration process is finished. You can then start to profile your
    application locally.
Remote Debugging in NetBeans

Any Java executable that can be run from the IDE can be debugged by it. This gives a developer the full range of NetBeans’ debugging power for working with standalone applications, applets, and client applications that are based on a variety of technologies. The NetBeans debugger can also attach to a Java process already running in a separate JVM, either on the same computer or on a remote machine. This gives a developer the same power for debugging remote applications and components running in Java-based server containers. Attaching to a remote JVM makes it possible to use breakpoints, conditionals, watches, and other debugging features with servlets, Enterprise JavaBeans, and RMI or CORBA server objects. Debugging remote objects without such features requires inserting a myriad of System.out.println( ) statements or other logging code. Using logging code for debugging is slow and inefficient, whereas using NetBeans is much more powerful.

Debugging Remotely

1. Start the JVM that you want to debug with the following switches:
2. java -Xint -Xdebug -Xnoagent -Xrunjdwp:transport=dt_socket,server=y,address=12999,suspend=n ...
   ○ -Xint: Turn off Hotspot optimizations. Although this is not strictly necessary, it does make the JVM far more stable while debugging.
   ○ -Xdebug: Turn on debug mode.
   ○ -Xnoagent: Disable the old sun.tools.debug.
   ○ -Xrunjdwp: Specify connection details:
      i. transport=dt_socket: tell the JVM to allow a remote debugger to connect via socket. Alternatively, transport=dt_shmem allows the remote debugger to connect over shared memory, which requires that the debugger be on the same machine as the JVM. As of NetBeans 3.2.1, it seems that only dt_socket is supported.
      ii. server=y: tell the JVM to wait for a remote debugger to attach to it. Alternatively, server=n would cause the JVM to try to attach to a remote debugger.
      iii. address=12999: specify port number to listen on. This can be any port that you have permissions to. Remember this port because you will need it to configure NetBeans later.
      iv. suspend=n: if this is “y,” it causes the JVM to suspend before the main class is loaded.
4. Select Debug → Attach from the menus.
5. In the Attach window select the Default Debugger, the host you are connecting to, and the port on that machine that you specified when you started the JVM.

![Attach window](https://www.safaribooksonline.com/library/view/netbeans-the-definitive/0596002807/ch04s08.html)

6. Use the NetBeans Debugger just as you would a local debugging session.

Source from, pictures taken from my local computer:
https://www.safaribooksonline.com/library/view/netbeans-the-definitive/0596002807/ch04s08.html