



# Beyond Hello World - Advanced Arm Compiler 5 Features

Version 1.0

## Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).  
All rights reserved.

## Issue 01

102748\_0100\_01\_en



## Beyond Hello World - Advanced Arm Compiler 5 Features

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

### Release information

#### Document history

Issue	Date	Confidentiality	Change
0100-01	12 November 2021	Non-Confidential	Initial release

### Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly

or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

## Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

## Product Status

The information in this document is Final, that is for a developed product.

## Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on <https://support.developer.arm.com>

To provide feedback on the document, fill the following survey: <https://developer.arm.com/documentation-feedback-survey>.

## Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email [terms@arm.com](mailto:terms@arm.com).

# Contents

1. Introduction.....	6
2. Compiling mixed C and assembly source files.....	7
3. Sharing header files between C and assembly code.....	10
4. Improving optimization with linker feedback.....	12
5. Further reading.....	17

# 1. Introduction

The [Building hello world using Arm Compiler](#) tutorial shows you how to build a simple C program with the Arm Compiler 5 toolchain.

This tutorial moves beyond the basics to explore some of the more advanced features of the Arm Compiler 5 toolchain.

This tutorial assumes you have installed and licensed Arm DS-5 Development Studio. For more information, see [Getting Started with Arm DS-5 Development Studio](#).

## 2. Compiling mixed C and assembly source files

The Arm assembler, `armasm` reads assembly language source code and outputs object code.

The Arm compiler, `armcc` compiles C and C++ source to object code.

The Arm linker, `armlink` combines the contents of one or more object files with any required libraries to produce an executable program.

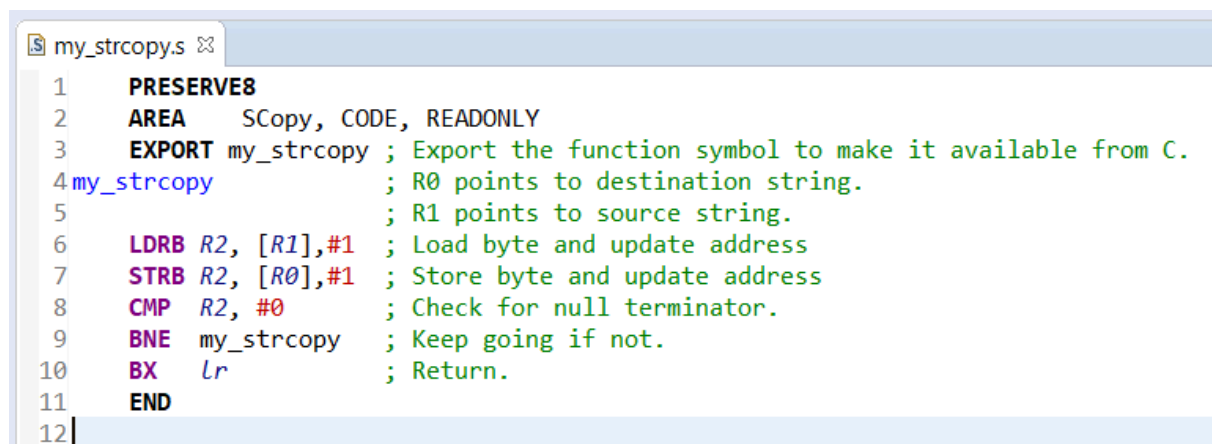
The following example shows how to use `armasm`, `armcc`, and `armlink` from DS-5 to build a project containing both C and assembly source files.

1. Create a new C project and add a new source file `my_strcopy.s` containing the following assembly code:

```
PRESERVE8
AREA      SCopy, CODE, READONLY
EXPORT my_strcopy ; Export symbol
my_strcopy ; R0 -> dest string
           ; R1 -> source string
    LDRB R2, [R1],#1 ; Load byte + update addr
    STRB R2, [R0],#1 ; Store byte + update addr
    CMP  R2, #0      ; Check for null
    BNE  my_strcopy  ; Keep going if not
    BX   lr          ; Return
END
```

The function `my_strcopy()` is exported so that it is available to be used from C.

**Figure 2-1: An image of an assembly source.**



2. Add a new source file to the project with the name `test.c` containing the following C code:

```
#include <stdio.h>

/* Declare the assembly function */
extern void my_strcopy(char *d, const char *s);
```

```

int main()
{
    const char *srcstr = "First string - source ";
    char dststr[] = "Second string - dest ";

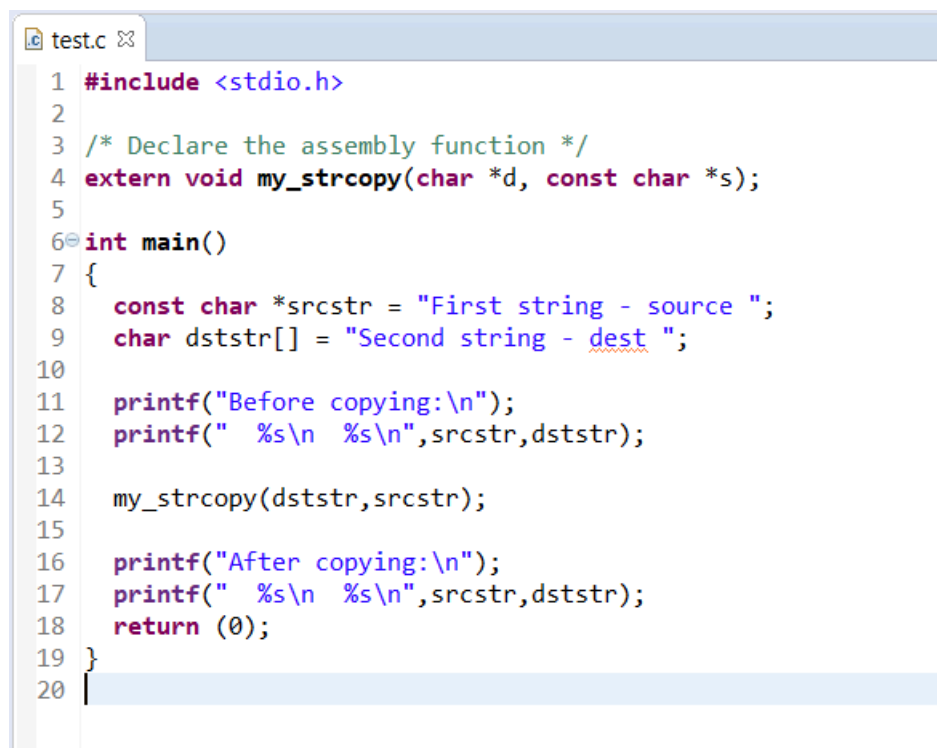
    printf("Before copying:\n");
    printf(" %s\n %s\n",srcstr,dststr);

    my_strcopy(dststr,srcstr);

    printf("After copying:\n");
    printf(" %s\n %s\n",srcstr,dststr);
    return (0);
}

```

**Figure 2-2: An image of a c source.**



```

test.c
1  #include <stdio.h>
2
3  /* Declare the assembly function */
4  extern void my_strcopy(char *d, const char *s);
5
6  int main()
7  {
8      const char *srcstr = "First string - source ";
9      char dststr[] = "Second string - dest ";
10
11     printf("Before copying:\n");
12     printf(" %s\n %s\n",srcstr,dststr);
13
14     my_strcopy(dststr,srcstr);
15
16     printf("After copying:\n");
17     printf(" %s\n %s\n",srcstr,dststr);
18     return (0);
19 }
20

```

3. Build the project. The Arm Compiler toolchain does the following:
  - a. Assembles my\_strcopy.s with armasm to produce the object file my\_strcopy.o.
  - b. Compiles test.c with armcc to produce the object file test.o.
  - c. Links the object files with armlink to produce an executable image.

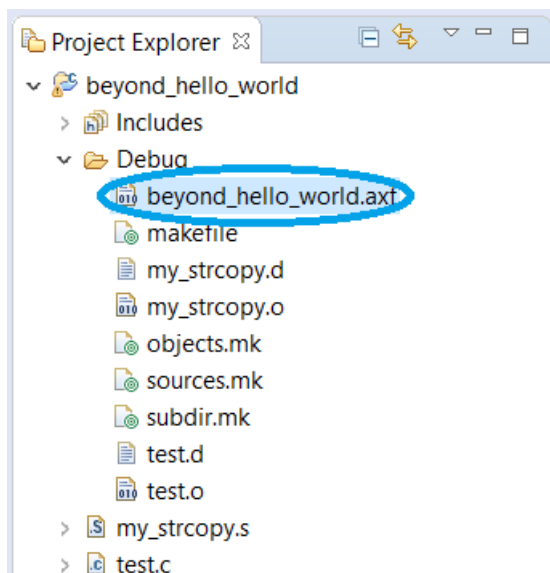
When you run the executable image, it produces the following output:

```

Before copying:
First string - source
Second string - dest
After copying:
First string - source
First string - source

```



**Figure 2-3: A compiled image.**

### 3. Sharing header files between C and assembly code

The usual way to define constants in C code is to use `#define`-s, or in assembly code to use `equ` directives. If your project contains a mixture of C and assembly code, there might be some constant definitions that are common to both. If so, to avoid maintaining two separate lists, you can create one list of common definitions and include them in both your C and assembly code.

To do this, you can use C-style `#include` and `#define` directives directly in your assembly source code. You can pass this source code through the `armcc` C preprocessor. This outputs a preprocessed version of your assembly code which `armasm` can then assemble.

The following example shows how to do this.

1. Add a header file called `my_strcopy.h` to the project, containing the following line:

```
#define ONE_CONSTANT 1
```

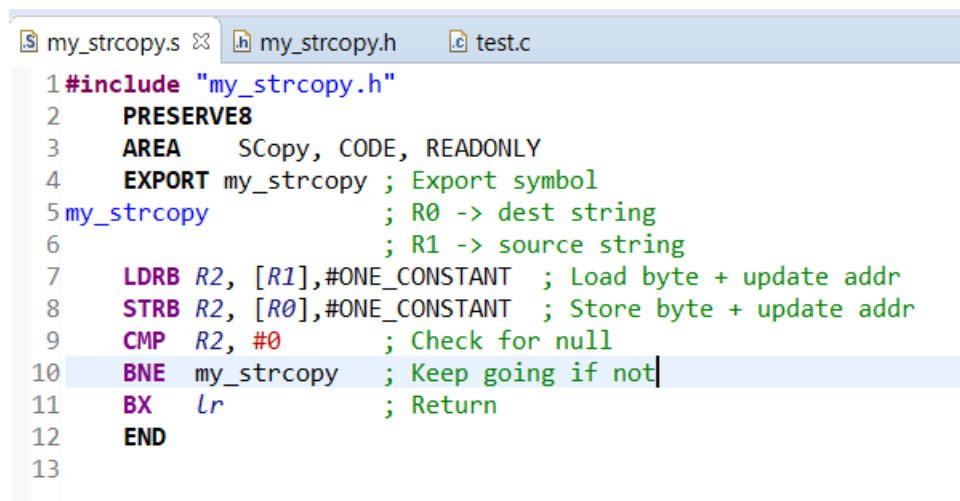
2. Add this line to the top of `my_strcopy.s`, created in the previous example:

```
#include "my_strcopy.h"
```

3. In `my_strcopy.s`, replace the occurrences of `#1` with `#ONE_CONSTANT`, for example:

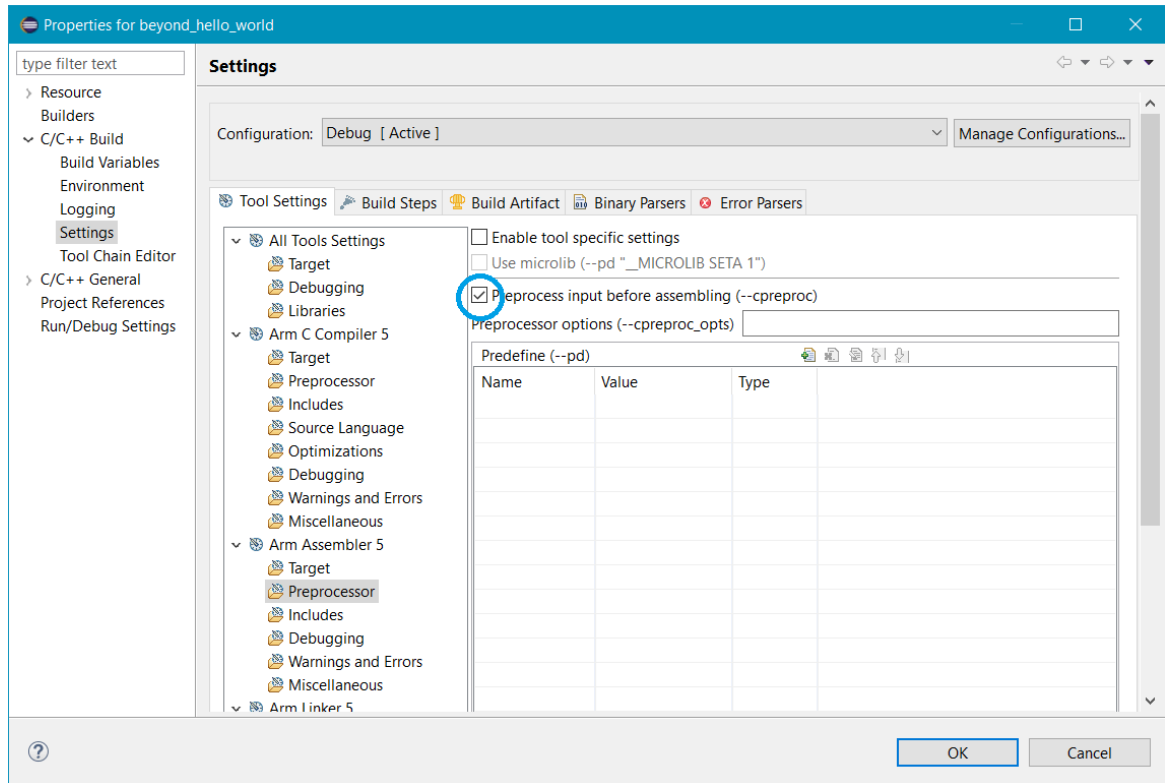
```
LDRB R2, [R1], #ONE_CONSTANT
```

**Figure 3-1: A image of an assembly source.**



4. Pass `my_strcopy.s` through the C preprocessor. If you tried to build the project without first doing this, `armasm` would report a syntax error for the `#include` statement you added to `my_strcopy.s`.
  - a. Open the Project Settings dialog.

- b. Under C/C++ build, select Settings.
- c. In the Tool Settings tab, under Arm Assembler 5, select Preprocessor.
- d. Tick the box marked `Preprocess input before assembling (--cpreproc)`.



`armasm` automatically passes some command-line options to the C preprocessor. If you need to pass other simple command-line options to the C preprocessor, for example `-D` or `-I`, specify them in the field marked `Preprocessor options (--cpreproc_opts)`.

## 4. Improving optimization with linker feedback

Linker feedback lets the compiler and linker collaborate to improve the removal of unused code.

The linker can produce a text file containing a list of unused functions and functions that have been inlined. This information can be fed back to the compiler, which rebuilds the objects, placing these functions in their own sections. These sections can then be removed by the linker during usual unused section elimination.

The following example shows how linker feedback works.

1. Create a new C project and add a new source file `fb.c` containing the following code:

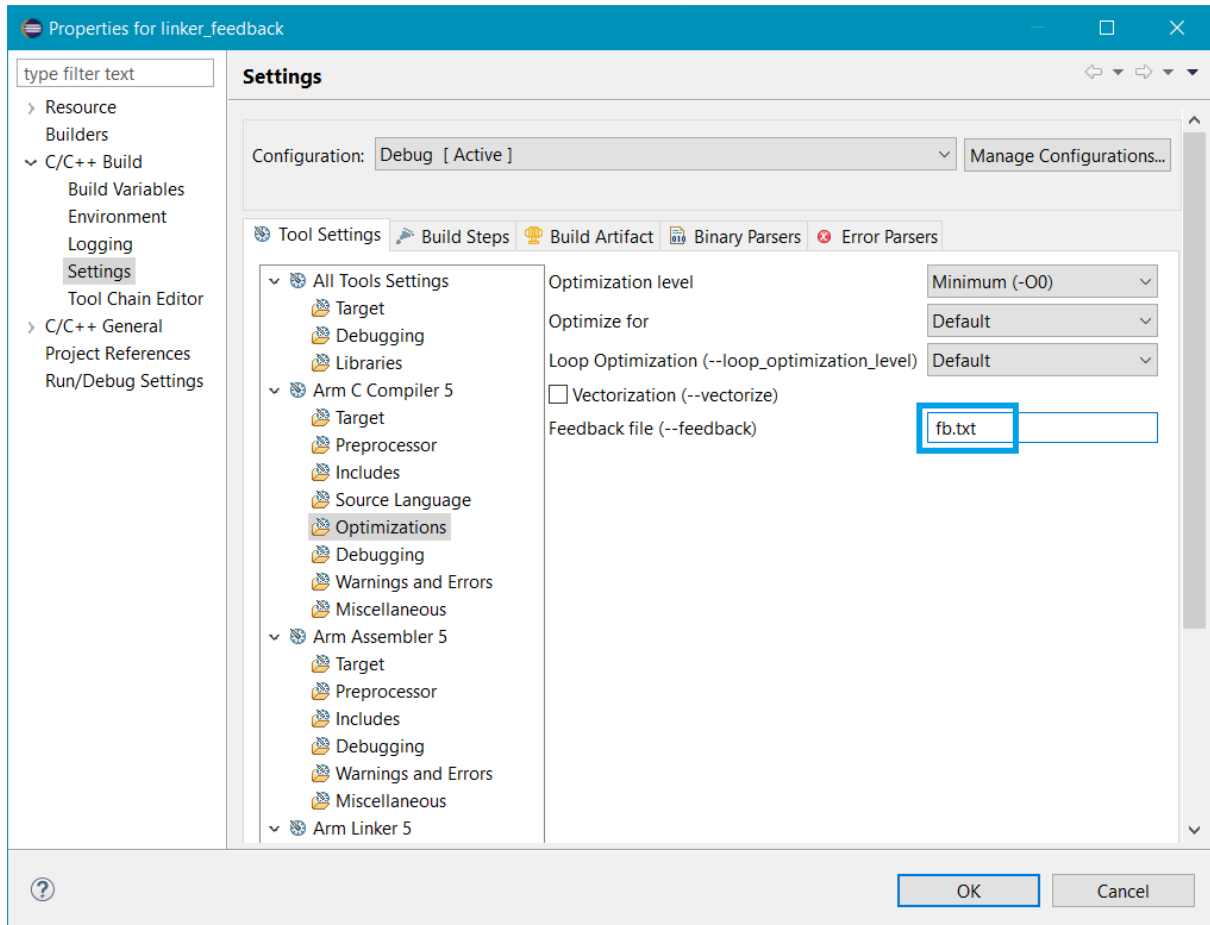
```
#include <stdio.h>
void legacy()
{
    printf("This is an unused function.\n");
}

int cubed(int i)
{
    return i*i*i;
}

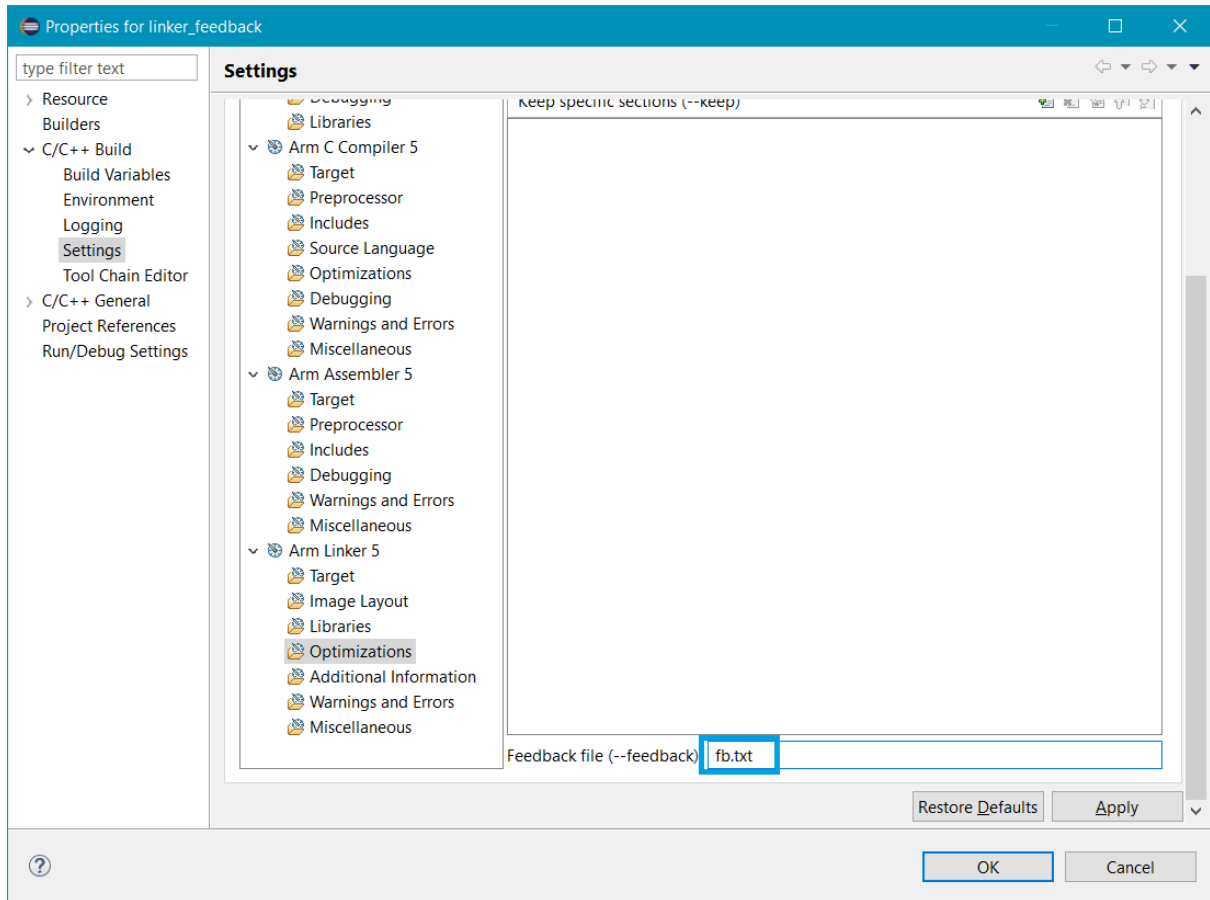
int main(void)
{
    int n = 3;
    printf("%d cubed = %d\n", n, cubed(n));
}
```

2. Open the Properties dialog box for your project, and select C/C++ Build > Settings.

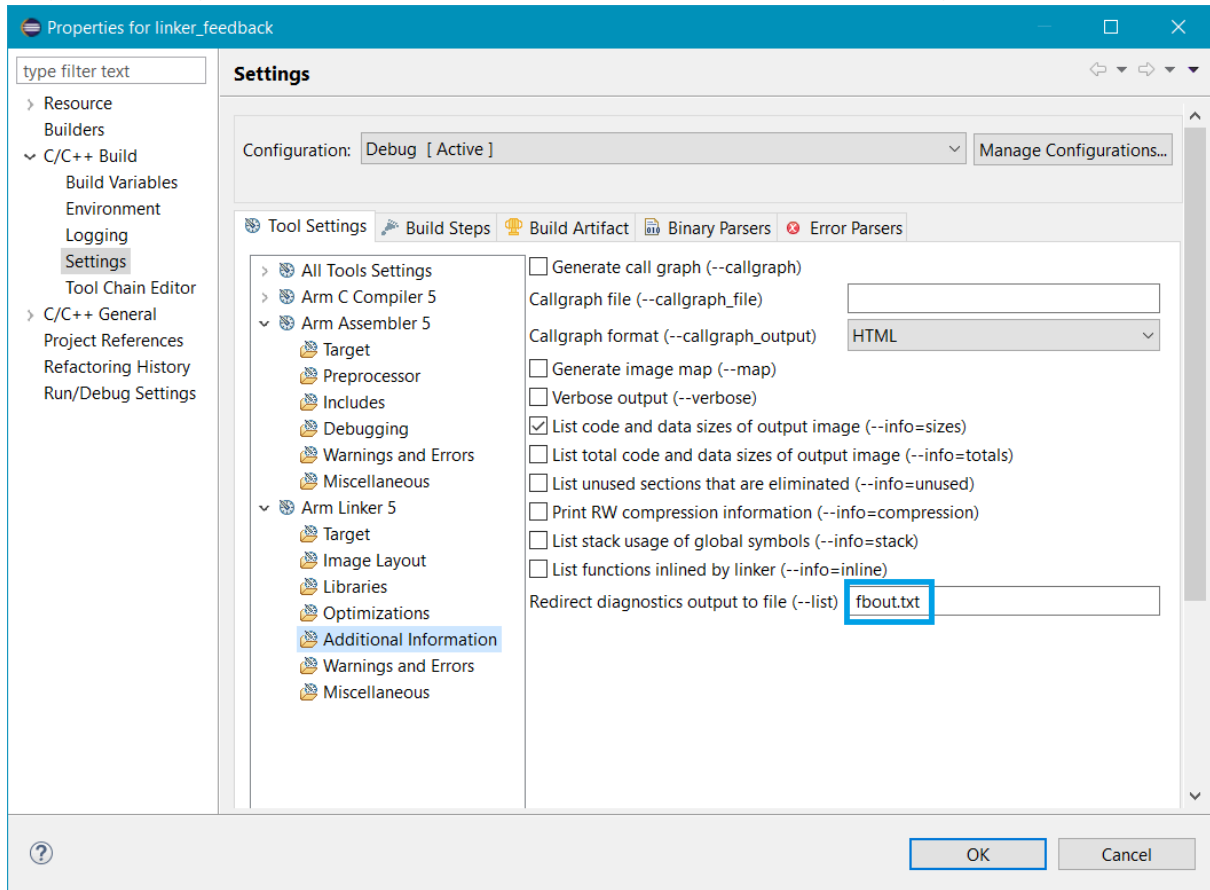
3. On the Settings tab, select Arm C Compiler 5 > Optimizations and specify “Feedback file (--feedback)” `fb.txt`. This tells the compiler to look for a feedback file.



4. On the Settings tab, select Arm Linker 5 > Optimizations and specify “Feedback file (--feedback)” `fb.txt`. This tells the linker to create a feedback file.



- On the Tool Settings tab, select Arm Linker 5 > Additional Information and specify “Redirect diagnostics output to file (--list)” `fbout.txt`. This tells the linker to save diagnostics to a file.



- Build the project. For this first compilation, the feedback file does not exist at compilation time. At link time, the linker identifies `legacy()` as the unused function, and creates a feedback file `fb.txt` containing this information.
- Clean the project (Project > Clean...), to remove the object and image files.
- Rename the diagnostics file `fbout.txt` to `fbout_orig.txt` to let you compare it to the next build.
- Build the project again. This time, the feedback file does exist at compilation time. Because the feedback file informs the compiler that `legacy()` is an unused function, the compiler can now omit this function from the generated object code.

You can compare the two diagnostics files, `fbout.txt` and `fbout_orig.txt`, to see the sizes of the image components (for example, Code, RO Data, RW Data, and ZI Data). The Code

component is smaller, because `armlink` has removed the `legacy()` function from the final image.

fbout_orig.txt								fbout.txt							
81	24	0	0	0	0	76	sys_wrch.o	81	24	0	0	0	0	76	sys_wrch.o
82	4	0	0	0	0	68	use_no_semi.o	82	4	0	0	0	0	68	use_no_semi.o
83								83							
84	-----							84	-----						
85	5632	244	12	16	348	3944	Library Totals	85	5632	244	12	16	348	3944	Library Totals
86	0	0	0	0	0	0	(incl. Padding)	86	0	0	0	0	0	0	(incl. Padding)
87	-----							87	-----						
88	-----							88	-----						
89								89							
90	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Library Name		90	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Library Name	
91								91							
92	5632	244	12	16	348	3944	c_4.l	92	5632	244	12	16	348	3944	c_4.l
93	-----							93	-----						
94								94							
95	5632	244	12	16	348	3944	Library Totals	95	5632	244	12	16	348	3944	Library Totals
96	-----							96	-----						
97	-----							97	-----						
98	-----							98	-----						
99	=====							99	=====						
100								100							
101								101							
102	Code (inc. data)	RO Data	RW Data	ZI Data	Debug			102	Code (inc. data)	RO Data	RW Data	ZI Data	Debug		
103								103							
104	5776	312	28	16	348	4040	Grand Totals	104	5708	260	28	16	348	3937	Grand Totals
105	5776	312	28	16	348	4040	ELF Image Totals	105	5708	260	28	16	348	3937	ELF Image Totals
106	5776	312	28	16	0	0	ROM Totals	106	5708	260	28	16	0	0	ROM Totals
107	=====							107	=====						
108	=====							108	=====						
109	=====							109	=====						
110	Total RO Size (Code + RO Data)					5804 ( 5.67kB)		110	Total RO Size (Code + RO Data)					5736 ( 5.60kB)	
111	Total RW Size (RW Data + ZI Data)					364 ( 0.36kB)		111	Total RW Size (RW Data + ZI Data)					364 ( 0.36kB)	
112	Total ROM Size (Code + RO Data + RW Data)					5820 ( 5.68kB)		112	Total ROM Size (Code + RO Data + RW Data)					5752 ( 5.62kB)	
113	=====							113	=====						
114	=====							114	=====						
115								115							



## 5. Further reading

To read more about Arm Compiler 5, follow the links:

- [-cpreproc](#)
- [-cpreproc\\_opts=option\[,option,...\]](#)
- [Using the C preprocessor](#)

Arm Compiler `armcc` User Guide

- [-feedback=filename](#)
- [About linker feedback](#)
- [Example of using linker feedback](#)
- [-feedback=filename](#)