

>



using z88dk the tool zcc to compile VMEX, I have this reported error:

```
..\src/tests.h::test::0::16:161: error: duplicate definition: _auto_test_testprog
^---- _auto_test_testprog
    ^---- _auto_test_testprog:
..\src/tests.h::test::0::16:161: error: duplicate definition: _auto_test_testprog
^---- _auto_test_testprog
    ^---- _auto_test_testprog:
..\src/tests.h::test::0::16:161: error: duplicate definition: _auto_test_testprog
^---- _auto_test_testprog
    ^---- _auto_test_testprog:
..\src/tests.h::test::0::16:161: error: duplicate definition: _auto_test_testprog
^---- _auto_test_testprog
    ^---- _auto_test_testprog:
..\src/tests.h::test::0::16:161: error: duplicate definition: _auto_test_testprog
```



The error message you're receiving indicates that there's a duplicate definition of the symbol **`_auto_test_testprog`** in your code. This usually happens when you include a header file multiple times or when the same symbol is defined in different files. To fix this issue, you can try the following steps:

1. Use include guards: Make sure that your header files have proper include guards to prevent multiple inclusions. Include guards are preprocessor directives that prevent the contents of the header file from being included more than once. Here's an example:

c

Copy code

```
#ifndef TESTS_H #define TESTS_H // Your header file content here #endif //
TESTS_H
```

Replace **`TESTS_H`** with a unique identifier for each header file. This will prevent the contents of the header file from being included multiple times.

2. Use **`static`**: If the **`_auto_test_testprog`** symbol is a function or variable that should have file scope, you can use the **`static`** keyword. This will make the symbol visible only within

the same source file:

c

 Copy code

```
static void _auto_test_testprog() { // Function definition }
```

3. Check for multiple definitions: Ensure that you don't have multiple definitions of the same symbol in different source files. If you do, consider renaming the symbols or placing them in different namespaces to avoid conflicts.

After making these changes, try to recompile your code with `zcc`. If the issue persists, carefully review your code and ensure you haven't missed any duplicate definitions.



it seems I have declared `TU8 testprog[]` each time in separate local `{}` block and this is NOT failing in 5 different C compilers cant be some switch/pragma to allow this?



>

It is possible that `z88dk` is more strict about certain aspects of the C language compared to other compilers, and that's why you're encountering this issue. While some compilers may allow duplicate declarations in separate local blocks without generating errors or warnings, it's generally a good idea to adhere to C language best practices to avoid potential issues when switching compilers or platforms.

To address this issue while still keeping the **`TU8 testprog[]`** declarations local to their respective blocks, you can enclose the declarations within separate functions. This will ensure that the variable is local to the function scope and won't generate a duplicate definition error:

c

 Copy code

```
void function1() { TU8 testprog[] = { /* initialization */ }; // Rest of the  
code in the block } void function2() { TU8 testprog[] = { /* initialization */  
}; // Rest of the code in the block }
```

Alternatively, if you must use separate local blocks within the same function, you can declare the **testprog** variable within each block using a different name:

c

 Copy code

```
void some_function() { { TU8 testprog1[] = { /* initialization */ }; // Rest of
the code in the block } { TU8 testprog2[] = { /* initialization */ }; // Rest of
the code in the block } }
```

While you could try to find a compiler flag or pragma to allow duplicate declarations, it's generally better to resolve the issue at the source code level to ensure maximum compatibility with different compilers and environments.



umm, now I have reported the same thing in fact using static

```
..\src\tests.h::test::0::16:161: error: duplicate definition: _st_test_testprog
```

```
^---- _st_test_testprog
```

```
^---- _st_test_testprog:
```



I apologize for the confusion earlier. Given the new error message, it seems that the issue persists even after using the **static** keyword. To further diagnose the issue, it would be helpful to have more information about the structure of your code, particularly how you include the header file and where the problematic declarations are located.

However, as a general suggestion, if the symbol **_st_test_testprog** is declared within a header file (tests.h), you should consider moving the actual definition of the symbol to a source file (e.g., tests.c) and only declare the symbol in the header file. This way, you can prevent multiple definitions while still making the symbol available to other source files that include the header file.

In tests.h:

c

 Copy code

```
#ifndef TESTS_H #define TESTS_H // Declare the symbol extern TU8
_st_test_testprog[]; #endif // TESTS_H
```

In tests.c:

c

 Copy code

```
#include "tests.h" // Define the symbol TU8 _st_test_testprog[] = { /*
initialization */ };
```

Then, you can include the tests.h header file in any source file where you need to use the **_st_test_testprog** symbol, and the linker will ensure that only one definition is used.

If you provide more information about the structure of your code and the location of the problematic declarations, I would be able to give more specific guidance on how to resolve this issue.



umm, its clear that zcc constructs local identifiers from function name and identifier ?? my code of tests, lots of them are basically organized this way and everywhere else its okay I dont want to change this much

```
case OP00_NOP: //TODO z88dk / zcc issue: testprog replaced by local identifier here
{
    static TU8 OP00_NOP_test[] =
    {
        0x00,0x00,0x00,0x00, //nop
    };
    memset(&core, 0, sizeof(core));
    testresult = vmex(OP00_NOP_test, sizeof(OP00_NOP_test));
    printf("%s: %d\n", testname, testresult); if (testresult!=0) return 1;
}
break;
```

```
#ifdef TEST_IMMEDIATE
case OPI11_ADDI:
{
    static TU8 testprog[] =
```

```

{
    0x11,0xAA,0x00,0x08, //addi r10 r10 $8
    0x11,0xAA,0x00,0x02, //addi r10 r10 $2
    0x11,0xAA,0xFF,0xF4, //addi r10 r10 -12
    0x11,0xAA,0x00,0x02, //addi r10 r10 2

    0x11,0xAA,0xF0,0x00, //addi r10 r10 $F000 (61440) = -4096
    0x11,0xAA,0x9c,0x40, //addi r10 r10 40000 ($9C40) = -25536
    0x11,0xAA,0x73,0xC0, //addi r10 r10 29632 ($73C0) = 29632

    0x11,0xAA,0x30,0x00, //addi r10 r10 $3000 (12288)
    0x11,0xAA,0x30,0x00, //addi r10 r10 $3000 (12288)
    0x11,0xAA,0x90,0x00, //addi r10 r10 -$7000 (= $9000 = -28672)
    0x11,0xAA,0x10,0x00, //addi r10 r10 $1000 (4096)
};
memset(&core, 0, sizeof(core));
testresult = vmex(testprog, sizeof(testprog));
printf("%s: %d\n", testname, testresult); if (testresult!=0) return 1;
}
break;

```

case OPI12_SUBI:

```

{
    static TU8 testprog[] =
    {
        0x11,0xA0,0x00,0x08, //addi r10 r0 $8
        0x11,0xAA,0x00,0x02, //addi r10 r10 $2
        0x12,0xAA,0x00,0x0C, //subi r10 r10 12
        0x11,0xAA,0x00,0x02, //addi r10 r10 2

        0x11,0xAA,0x30,0x00, //addi r10 r10 $3000 (12288)
        0x11,0xAA,0x30,0x00, //addi r10 r10 $3000 (12288)
        0x12,0xAA,0x70,0x00, //subi r10 r10 $7000 //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        0x11,0xAA,0x10,0x00, //addi r10 r10 $1000 (4096)
    };
    memset(&core, 0, sizeof(core));
    testresult = vmex(testprog, sizeof(testprog));
    printf("%s: %d\n", testname, testresult); if (testresult!=0) return 1;
}

```

break;



I understand that you don't want to change the structure of your code too much, and it's working with other compilers. In this case, you can try using an anonymous union to create unique namespaces for each test within the same function. This way, you can keep the same variable name (**testprog**) for each test while preventing duplicate definition errors.

Here's how you can modify your code using anonymous unions:

c

Copy code

```
case OP00_NOP: //TODO z88dk / zcc issue: testprog replaced by local identifier
here { union { struct { TU8 testprog[]; }; } = { { 0x00,0x00,0x00,0x00, //nop }
}; memset(&core, 0, sizeof(core)); testresult = vmex(testprog,
sizeof(testprog)); printf("%s: %d\n", testname, testresult); if (testresult!=0)
return 1; } break; #ifdef TEST_IMMEDIATE case OPI11_ADDI: { union { struct { TU8
testprog[]; }; } = { { 0x11,0xAA,0x00,0x08, //addi r10 r10 $8 // ... (rest of
the testprog values) } }; memset(&core, 0, sizeof(core)); testresult =
vmex(testprog, sizeof(testprog)); printf("%s: %d\n", testname, testresult); if
(testresult!=0) return 1; } break; case OPI12_SUBI: { union { struct { TU8
testprog[]; }; } = { { 0x11,0xA0,0x00,0x08, //addi r10 r0 $8 // ... (rest of the
testprog values) } }; memset(&core, 0, sizeof(core)); testresult =
vmex(testprog, sizeof(testprog)); printf("%s: %d\n", testname, testresult); if
(testresult!=0) return 1; } break;
```

Using anonymous unions with nested structs will create unique namespaces for each test case, allowing you to use the same **testprog** variable name without causing duplicate definition errors.

Please note that anonymous unions are supported by C11 and later standards, so make sure your compiler supports C11 or later. If z88dk doesn't support anonymous unions by default, you may need to provide a compiler flag to enable C11 support (e.g., **-std=c11**).



ya, thanks, sounds possible, more simple way isnt possible? and probably not, ZCC will not support C11 it seems, its old



Given the limitations of the z88dk compiler and the fact that it does not support C11, we can try another simple method using the preprocessor to give unique names to the **testprog** arrays for each test case.

You can define a macro that concatenates a prefix with the test case identifier, like this:

c

Copy code

```
#define TESTPROG_NAME(name) testprog_ ## name
```

Now, you can use this macro to create unique names for each **testprog** array:

c

Copy code

```
case OP00_NOP: { static TU8 TESTPROG_NAME(OP00_NOP)[] = { 0x00,0x00,0x00,0x00,
//nop }; memset(&core, 0, sizeof(core)); testresult =
vmex(TESTPROG_NAME(OP00_NOP), sizeof(TESTPROG_NAME(OP00_NOP))); printf("%s:
%d\n", testname, testresult); if (testresult!=0) return 1; } break; #ifdef
TEST_IMMEDIATE case OPI11_ADDI: { static TU8 TESTPROG_NAME(OPI11_ADDI)[] = {
0x11,0xAA,0x00,0x08, //addi r10 r10 $8 // ... (rest of the testprog values) };
memset(&core, 0, sizeof(core)); testresult = vmex(TESTPROG_NAME(OPI11_ADDI),
sizeof(TESTPROG_NAME(OPI11_ADDI))); printf("%s: %d\n", testname, testresult); if
(testresult!=0) return 1; } break; case OPI12_SUBI: { static
```



well in this case, I can prepend the testprog by case name simply, ya, will see



Yes, you can simply prepend the **testprog** variable with the case name to create unique names for each test case. This way, you can avoid duplicate definition errors while keeping the code relatively simple and compatible with the z88dk compiler.

Here's an example of how you can modify your code:

```
case OP00_NOP: { static TU8 OP00_NOP_testprog[] = { 0x00,0x00,0x00,0x00, //nop
}; memset(&core, 0, sizeof(core)); testresult = vmex(OP00_NOP_testprog,
sizeof(OP00_NOP_testprog)); printf("%s: %d\n", testname, testresult); if
(testresult!=0) return 1; } break; #ifdef TEST_IMMEDIATE case OPI11_ADDI: {
static TU8 OPI11_ADDI_testprog[] = { 0x11,0xAA,0x00,0x08, //addi r10 r10 $8 //
... (rest of the testprog values) }; memset(&core, 0, sizeof(core)); testresult
= vmex(OPI11_ADDI_testprog, sizeof(OPI11_ADDI_testprog)); printf("%s: %d\n",
testname, testresult); if (testresult!=0) return 1; } break; case OPI12_SUBI: {
static TU8 OPI12_SUBI_testprog[] = { 0x11,0xA0,0x00,0x08, //addi r10 r0 $8 //
... (rest of the testprog values) }; memset(&core, 0, sizeof(core)); testresult
= vmex(OPI12_SUBI_testprog, sizeof(OPI12_SUBI_testprog)); printf("%s: %d\n",
testname, testresult); if (testresult!=0) return 1; } break;
```

By using unique names for each **testprog** array, you should be able to resolve the duplicate definition issue while maintaining compatibility with the z88dk compiler.



ya, thanks, this sounds as simplest way, ufffff



You're welcome! I'm glad that the simple solution meets your requirements. Using unique names for each **testprog** array should help you avoid the duplicate definition issue while keeping your code compatible with the z88dk compiler.

If you encounter any other issues or have questions, please don't hesitate to ask. I'm here to help. Good luck with your project!



so far so good, this helped with z88dk now compiling, community helped with cc65 now compiling and running on 4 platforms I need most - so thank YOU, GPT4 :-)



You're welcome! I'm glad to hear that the solution worked for you and that your code is now compiling and running on the platforms you need. It's great to know that the

community has been helpful as well.

If you have any more questions or need further assistance in the future, please don't hesitate to ask. Good luck with your project, and happy coding!
