Why do I get errors about some weird symbol called ?main@@YAHP$01E$AAV?$Array@PE$AAVString@Platform…, part 1

devblogs.microsoft.com/oldnewthing/20250625-00/?p=111306

June 25, 2025

A colleague was writing a fuzz test and ran into a build error.

```
vccorlib.lib(climain.obj) : error LNK2019: unresolved external symbol "?
main@@YAHP$01E$AAV?$Array@PE$AAVString@Platform@@$00@Platform@@@Z" (?
main@@YAHP$01E$AAV?$Array@PE$AAVString@Platform@@$00@Platform@@@Z) referenced in
function "int __cdecl _main(void)" (?_main@@YAHXZ)
```

What does this mean?

My colleague was writing a fuzz test in C++, but noted that other parts of the component are written in C++/CX.

I could have guessed that C++/CX was involved because the missing symbol says that it's a function named `main` which takes parameters that involve things named `Array`, `String`, and `Platform`.

The signature for the `main` function in a C++/CX program is

```
int main(Platform::Array<Platform::String^>^ args)
```

so that seems to match up with the words we picked out of the decorated name.

(Amusingly, C++/CX is such a black sheep that the linker's decorated name decoder can't even decode C++/CX names.)

The usage of this particular fuzz test library is similar to LLVM's libFuzzer: You define a fuzzing entrypoint that takes a memory block, and the library provides a `main` function that calls your function repeatedly with different blocks of memory.

We pulled in the Visual Studio team to help figure out why the wrong `main` function was being requested.

They suggested linking with the `/verbose` to get more insight into how the linker is working.

```
      Found mainCRTStartup
        Loaded libcmt.lib(exe_main.obj)

      Found main
        Referenced in LIBCMT.lib(exe_main.obj)
        Loaded vccorlib.lib(main.obj)

      Found "int __cdecl _main(void)" (?_main@@YAHXZ)
        Referenced in vccorlib.lib(main.obj)
        Loaded vccorlib.lib(climain.obj)

vccorlib.lib(climain.obj) : error LNK2019: unresolved external symbol "?
main@@YAHP$01E$AAV?$Array@PE$AAVString@Platform@@$00@Platform@@@Z" (?
main@@YAHP$01E$AAV?$Array@PE$AAVString@Platform@@$00@Platform@@@Z) referenced in
function "int __cdecl _main(void)" (?_main@@YAHXZ)
```

The error message says that `vccorlib.lib`'s `climain.obj` has a reference to the weird version of `main`. Working backward through the verbose output, we see that `vccorlib.lib(climain.obj)` was in turn added to the binary because it satisfied a search for `int __cdecl _main(void)` that was requested by `vccorlib.lib(main.obj)`.

The `vccorlib.lib(main.obj)` was added to the binary because it satisfied a request for `main` from `libcmt.lib(exe_main.obj)`.

And `libcmt.lib(exe_main.obj)`. was added to the binary because it satisfied a request for `mainCRTStartup`, which was presumably requested by the linker because this was linked as a console program.

The problem is that the `main` function was found in `vccorlib.lib` rather than in the fuzzer library.

The final piece of the puzzle is [the linker documentation on the finer points of how it resolves symbols]:

> Object files on the command line are processed in the order they appear on the command line. Libraries are searched in command line order as well, with the following caveat: Symbols that are unresolved when bringing in an object file from a library are searched for in that library first, and then the following libraries from the command line and `/DEFAULTLIB` (Specify default library) directives, and then to any libraries at the beginning of the command line.

Okay, now we can piece the story together by working forward.

The linker starts by looking for `mainCRTStartup`, and that leads to the `main`, which in turn leads to the wrong `int __cdecl _main()`. We get the wrong one due to the "symbols that are unresolved when bringing in an object file from a library" rule: The linker got `main` from `vccorlib.lib`, so the search for `_main` begins at `vccorlib.lib`, and that's why it finds the one in `vccorlib.lib(climain.obj)` instead of the one in the fuzzer library.

Now that we understand what happened, we can produce a minimal reproducible example.

```
>lib.cpp echo void fuzzme(); int __cdecl main(int, char**) { fuzzme(); return 42;
}
cl /c lib.cpp
lib /out:lib.lib lib.obj
```

This first step creates a library which provides a definition of `main`. This is the minimal version of the fuzzer library.

```
>fuzzer.cpp echo void fuzzme() {}
cl /c fuzzer.cpp
```

The next step creates the fuzzer client that the fuzzer library calls.

And then we can try to link it and see what happens.

```
link /out:fuzzer.exe /subsystem:console fuzzer.obj lib.lib

rem succeeds!
```

Hm, our attempt to create a minimal reproduction failed. There must be something we are missing.

We'll continue the investigation next time.