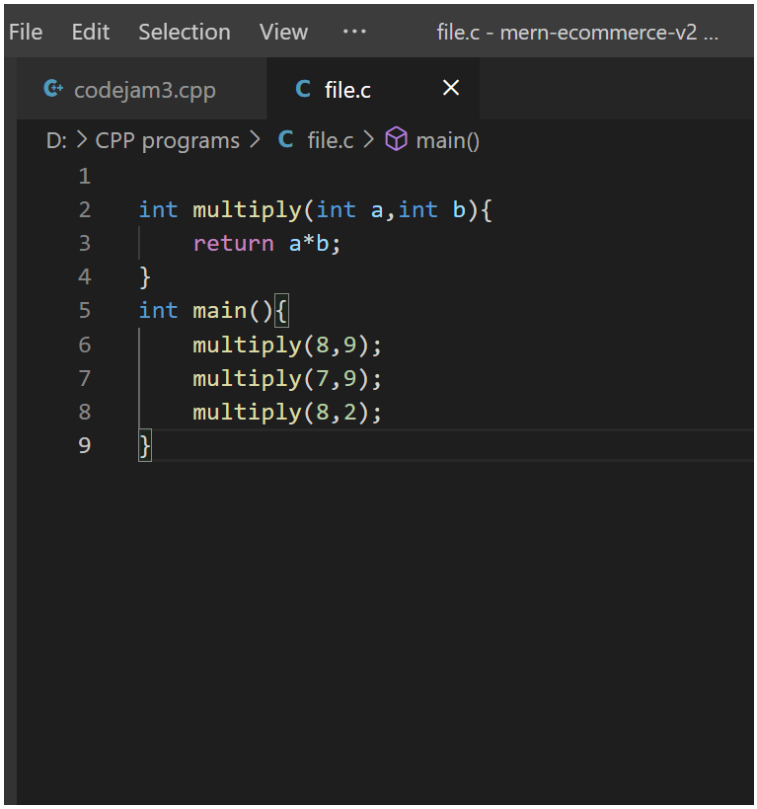
STAGES OF COMPILER

There are 4 stages of compiler design.

SOURCE CODE:



1) **Preprocessing**

   This stage opens all the header files and copy code in the file. The output of this stage is .i file.

   Command: gcc -E file.c

```
Command Prompt

D:\CPP programs>gcc -E file.c
# 1 "file.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "file.c"

int multiply(int a,int b){
    return a*b;
}
int main(){
    multiply(8,9);
    multiply(7,9);
    multiply(8,2);
}

D:\CPP programs>
```

2) **<u>Compilation</u>**

This stage generated assembly code from preprocessed .i file. The output of this stage is .s file.

<u>Command</u>: gcc -S file.c

```
Command Prompt
D:\CPP programs>gcc -S file.c

D:\CPP programs>type file.s
        .file   "file.c"
        .text
        .globl  multiply
        .def    multiply;       .scl    2;      .type   32;     .endef
        .seh_proc       multiply
multiply:
        pushq   %rbp
        .seh_pushreg    %rbp
        movq    %rsp, %rbp
        .seh_setframe   %rbp, 0
        .seh_endprologue
        movl    %ecx, 16(%rbp)
        movl    %edx, 24(%rbp)
        movl    16(%rbp), %eax
        imull   24(%rbp), %eax
        popq    %rbp
        ret
        .seh_endproc
        .def    __main; .scl    2;      .type   32;     .endef
        .globl  main
        .def    main;   .scl    2;      .type   32;     .endef
        .seh_proc       main
main:
        pushq   %rbp
        .seh_pushreg    %rbp
        movq    %rsp, %rbp
        .seh_setframe   %rbp, 0
        subq    $32, %rsp
        .seh_stackalloc 32
        .seh_endprologue
        call    __main
        movl    $9, %edx
        movl    $8, %ecx
        call    multiply
        movl    $9, %edx
        movl    $7, %ecx
        call    multiply
        movl    $2, %edx
        movl    $8, %ecx
        call    multiply
        movl    $0, %eax
        addq    $32, %rsp
        popq    %rbp
        ret
        .seh_endproc
        .ident  "GCC: (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0"
```

3) Assemble

This stage generates object files from assembly code. The output of this stage is .o file.

Command: gcc -c file.c

```
Command Prompt
D:\CPP programs>gcc -c file.c

D:\CPP programs>objdump -D file.o

file.o:     file format pe-x86-64


Disassembly of section .text:

0000000000000000 <multiply>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 4d 10                mov    %ecx,0x10(%rbp)
   7:   89 55 18                mov    %edx,0x18(%rbp)
   a:   8b 45 10                mov    0x10(%rbp),%eax
   d:   0f af 45 18             imul   0x18(%rbp),%eax
  11:   5d                      pop    %rbp
  12:   c3                      retq

0000000000000013 <main>:
  13:   55                      push   %rbp
  14:   48 89 e5                mov    %rsp,%rbp
  17:   48 83 ec 20             sub    $0x20,%rsp
  1b:   e8 00 00 00 00          callq  20 <main+0xd>
  20:   ba 09 00 00 00          mov    $0x9,%edx
  25:   b9 08 00 00 00          mov    $0x8,%ecx
  2a:   e8 d1 ff ff ff          callq  0 <multiply>
  2f:   ba 09 00 00 00          mov    $0x9,%edx
  34:   b9 07 00 00 00          mov    $0x7,%ecx
  39:   e8 c2 ff ff ff          callq  0 <multiply>
  3e:   ba 02 00 00 00          mov    $0x2,%edx
  43:   b9 08 00 00 00          mov    $0x8,%ecx
  48:   e8 b3 ff ff ff          callq  0 <multiply>
  4d:   b8 00 00 00 00          mov    $0x0,%eax
  52:   48 83 c4 20             add    $0x20,%rsp
  56:   5d                      pop    %rbp
  57:   c3                      retq
  58:   90                      nop
  59:   90                      nop
  5a:   90                      nop
  5b:   90                      nop
  5c:   90                      nop
  5d:   90                      nop
  5e:   90                      nop
  5f:   90                      nop

Disassembly of section .xdata:

0000000000000000 <.xdata>:
   0:   01 04 02                add    %eax,(%rdx,%rax,1)
```

4) Linking

This stage links all the different object files of a project also internal linking of functions is done in this file. The output is .exe file.

Command: gcc file.c -o file.exe

Command Prompt

```
D:\CPP programs>gcc file.c -o file.exe

D:\CPP programs>objdump -D file.exe

file.exe:     file format pei-x86-64


Disassembly of section .text:

0000000000401000 <__mingw_invalidParameterHandler>:
  401000:       c3                      retq
  401001:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  401006:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  40100d:       00 00 00

0000000000401010 <pre_c_init>:
  401010:       48 83 ec 28             sub    $0x28,%rsp
  401014:       48 8b 05 45 34 00 00    mov    0x3445(%rip),%rax        # 404460 <.refptr.mingw_initltsdrot_force>
  40101b:       31 d2                   xor    %edx,%edx
  40101d:       c7 00 01 00 00 00       movl   $0x1,(%rax)
  401023:       48 8b 05 46 34 00 00    mov    0x3446(%rip),%rax        # 404470 <.refptr.mingw_initltsdyn_force>
  40102a:       c7 00 01 00 00 00       movl   $0x1,(%rax)
  401030:       48 8b 05 49 34 00 00    mov    0x3449(%rip),%rax        # 404480 <.refptr.mingw_initltssuo_force>
  401037:       c7 00 01 00 00 00       movl   $0x1,(%rax)
  40103d:       48 8b 05 0c 34 00 00    mov    0x340c(%rip),%rax        # 404450 <.refptr.mingw_initcharmax>
  401044:       c7 00 01 00 00 00       movl   $0x1,(%rax)
  40104a:       48 8b 05 ef 32 00 00    mov    0x32ef(%rip),%rax        # 404340 <.refptr.__image_base__>
  401051:       66 81 38 4d 5a          cmpw   $0x5a4d,(%rax)
  401056:       74 58                   je     4010b0 <pre_c_init+0xa0>
  401058:       48 8b 05 e1 33 00 00    mov    0x33e1(%rip),%rax        # 404440 <.refptr.mingw_app_type>
  40105f:       89 15 a3 5f 00 00       mov    %edx,0x5fa3(%rip)        # 407008 <managedapp>
  401065:       8b 00                   mov    (%rax),%eax
  401067:       85 c0                   test   %eax,%eax
  401069:       74 35                   je     4010a0 <pre_c_init+0x90>
  40106b:       b9 02 00 00 00          mov    $0x2,%ecx
  401070:       e8 83 1a 00 00          callq  402af8 <__set_app_type>
  401075:       e8 f6 1a 00 00          callq  402b70 <__p__fmode>
  40107a:       48 8b 15 7f 33 00 00    mov    0x337f(%rip),%rdx        # 404400 <.refptr._fmode>
  401081:       8b 12                   mov    (%rdx),%edx
  401083:       89 10                   mov    %edx,(%rax)
  401085:       e8 06 06 00 00          callq  401690 <_setargv>
  40108a:       48 8b 05 5f 32 00 00    mov    0x325f(%rip),%rax        # 4042f0 <.refptr._MINGW_INSTALL_DEBUG_MATHERR>
  401091:       83 38 01                cmpl   $0x1,(%rax)
  401094:       74 5a                   je     4010f0 <pre_c_init+0xe0>
  401096:       31 c0                   xor    %eax,%eax
  401098:       48 83 c4 28             add    $0x28,%rsp
  40109c:       c3                      retq
  40109d:       0f 1f 00                nopl   (%rax)
  4010a0:       b9 01 00 00 00          mov    $0x1,%ecx
  4010a5:       e8 4e 1a 00 00          callq  402af8 <__set_app_type>
```

BOOTSTRAPPING

It is a technique of producing compiler for language X in X language. It sounds like a chicken- egg problem, but it can be developed with the help of already available languages. As we can first make a compiler for language X in language Y (for which compiler is already available) then we can write compiler for language X in language X and compile it using the compiler we made for the language. This way compiler for language X can be made in language X.