ABC by Example:
Translation Units and Header Files

"How to Avoid Putting Everything into a Single Source File"

all_in_one.abc

```
@ <stdio.hdr>

global foo: int = 42;

fn bar(): int
{
    return 13;
}

fn main()
{
    printf("foo = %d\n", foo);
    printf("bar() = %d\n", bar());
}
```

foobar.abc

```
global foo: int = 42;

fn bar(): int
{
    return 13;
}
```

xtest_foobar.abc

```
@ <stdio.hdr>


fn main()
{
    printf("foo = %d\n", foo);
    printf("bar() = %d\n", bar());
}
```

foobar.hdr

```
extern foo: int;
extern fn bar(): int;
```

foobar.abc

```
@ "foobar.hdr"

global foo: int = 42;

fn bar(): int
{
    return 13;
}
```

xtest_foobar.abc

```
@ <stdio.hdr>
@ "foobar.hdr"

fn main()
{
    printf("foo = %d\n", foo);
    printf("bar() = %d\n", bar());
}
```

foobar.hdr

```
extern foo: int;
extern fn bar(): int;
```

```
MCL:hpc0-abc lehn$ abc foobar.abc xtest_foobar.abc
MCL:hpc0-abc lehn$ ./a.out
foo = 42
bar() = 13
foo = 24
bar() = 13
```

foobar.abc

```
@ "foobar.hdr"

global foo: int = 42;

fn dummy()
{
    foo = 24;
}

fn bar(): int
{
    dummy();
    return 13;
}
```

xtest_foobar.abc

```
@ <stdio.hdr>
@ "foobar.hdr"

fn dummy()
{
    printf("foo = %d\n", foo);
    printf("bar() = %d\n", bar());
}

fn main()
{
    dummy();
    dummy();
}
```

foobar.hdr

```
extern foo: int;
extern fn bar(): int;
extern fn dummy();
```

```
MCL:hpc0-abc lehn$ abc foobar.abc xtest_foobar.abc
duplicate symbol '_dummy' in:
    /private/var/folders/9c/c0td6d_x0wd2z7wrql_m60z00000gn/T/xtest_foobar.o
    /private/var/folders/9c/c0td6d_x0wd2z7wrql_m60z00000gn/T/foobar.o
ld: 1 duplicate symbols
```

foobar.abc

```
@ "foobar.hdr"

global foo: int = 42;

fn dummy()
{
    foo = 24;
}

fn bar(): int
{
    dummy();
    return 13;
}
```

xtest_foobar.abc

```
@ <stdio.hdr>
@ "foobar.hdr"

fn dummy()
{
    printf("foo = %d\n", foo);
    printf("bar() = %d\n", bar());
}

fn main()
{
    dummy();
    dummy();
}
```

lexer.abc

```
@ <stdio.hdr>

enum TokenKind
{
    BAD_TOKEN,
    EOI,
    DECIMAL_LITERAL,
    PLUS,
};

global token: TokenKind = BAD_TOKEN;

fn getToken(): TokenKind;

// -- lexer implementation

global ch: int;

fn getToken(): TokenKind
{
    // ...
}

// -- simple test

fn main()
{
    // ...
}
```

lexer.abc

```
@ <stdio.hdr>
@ "lexer.hdr"

// -- lexer implementation
global token: TokenKind = BAD_TOKEN;

global ch: int;

fn getToken(): TokenKind
{
    while (ch == 0 || ch == ' ' || ch == '\n' || ch == '\t') {
        ch = getchar();
    }
    if (ch >= '0' && ch <= '9') {
        while (ch >= '0' && ch <= '9') {
            ch = getchar();
        }
        return token = DECIMAL_LITERAL;
    } else if (ch == '+') {
        ch = getchar();
        return token = PLUS;
    } else if (ch == EOF) {
        return token = EOI;
    } else {
        ch = getchar();
        return token = BAD_TOKEN;
    }
}
```

lexer.hdr

```
enum TokenKind
{
    BAD_TOKEN,
    EOI,
    DECIMAL_LITERAL,
    PLUS,
};

extern token: TokenKind;

extern fn getToken(): TokenKind;
```

xtest_lexer.abc

```
@ <stdio.hdr>
@ "lexer.hdr"

// -- simple test

fn main()
{
    while (getToken() != EOI) {
        if (token == DECIMAL_LITERAL) {
            printf("DECIMAL_LITERAL\n");
        } else if (token == BAD_TOKEN) {
            printf("BAD_TOKEN\n");
        } else if (token == PLUS) {
            printf("PLUS\n");
        } else {
            printf("unknown token: token = %d\n", token);
        }
    }
}
```