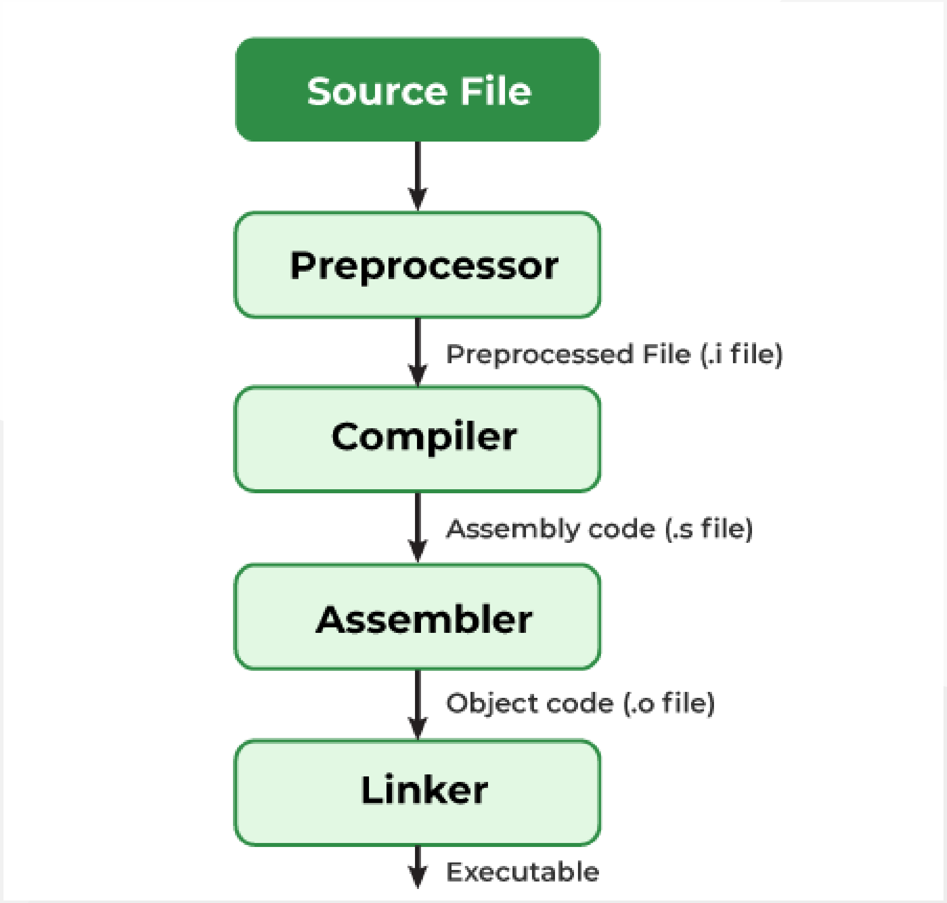
C Preprocesser, Compiler and Libraries



C Programming

- Procedural programming language
- Provides low-level memory access
- Requires explicit memory management
- Offers precise control over hardware
- Useful for embedded systems
- Standardized language usually cross-platform
- Long time use spanning many industries

Structure of a C Program

- Comments use `/* */` or `//`
- Functions defined with types
- Main is entry point
- Headers declare functions .h extension
- Must return int from main
- C source code files end with .c extension

Source Code

- Using a text editor or VS Code to write C code

```c
#include <stdio.h>

// Comments are helpful to understand your code
#define PI 3.14

/* Your program will use
   main() as well */

int main()
{
  printf("Hello World!");

  return 0;
}
```

C Preprocessor

- Preprocessing source code before compilation
- Output file has .i extension
- `#include` to add headers

```c
// Inserts code from .h into the source code file
#include <stdio.h>
#include "remake.h"

// Creates a symbolic name or constant expression
#define PI 3.14

/* Convert degrees to radians */
#define Deg_to_Rad(X) (X*M_PI/180.0)
```

C Preprocessor

- `#define`
  - Symolic name
  - Constant
  - Macro
  - `#define begin = {` and `#define end = }`

```
#define max(A,B) ( (A) > (B) ? (A):(B))
// in program
x = max(q+r,s+t); // uses defined max function

#define Deg_to_Rad(X) (X*M_PI/180.0)
#define LEFT_SHIFT_8 <<8
```

C Preprocessor

- `#if` / `#endif` / `#else` / `#elif`
    - Evaluates a constant integer expression
    - Always need `#endif`
    - `#ifdef` -- if defined and `#ifndef` -- if not defined

```
#ifdef OS_MSDOS
  #include <msdos.h>
#elifdef OS_UNIX
  #include "default.h"
#else
  #error Wrong OS!!
#endif
```

C Preprocessor Options

- -D control values set or defined from command line
  - `gcc -DLINELENGTH=80 prog.c -o prog` same as `#define LINELENGTH 80`
  - `#define` or `#undef` in program overrides
- -E force the compiler to stop after preprocessing stage

```
// gcc -DDEBUG prog.c -o prog
#ifdef DEBUG
   print("Debugging: Program Version 1\");
#else
   print("Program Version 1 (Production)\");
#endif
```

Preprocessor

- Output file has .i extension

Compiler

- Converts .i file to assembly code with .s extension
- Find Syntax Errors

Assembler

- Creates object code with .o extension

Linking

- Links object code with the libraries
- Creates an executable file

Common Library

- `stdio.h` - standard input/output
  - `printf(), scanf(), getchar()`
- `math.h` - mathematical functions
  - `sin(), cos(), tan(), pow(), sqrt()`
- `string.h` - string handling
  - `strlen(), strcpy(), strcat()`
- `file.h` - file handling
  - `fopen(), fclose(), fprintf(), fscanf()`
- `stdlib.h` - utility functions
  - `malloc(), calloc(), realloc(), srand()`

C Compiler

- Converts C code into an executable
- Popular compilers: GCC, Clang, Visual C++
  - Rutgers is using gcc10

```
gcc main.c                # Compiles the source code
./a.out
gcc main.c -o main        # Named the executable main
./main
gcc main.c -c             # Creates an object file
gcc main.o -o main        # Links the object file
./main
gcc main.c -O main        # Optimizes the executable
./main
gcc main.c -Wall main     # Displays all warnings
./main
```

Errors and Warnings

- Warnings indicate potential mistakes in code
- Line numbers with errors can narrow down
- Suggestions may be provided
- Syntax Errors
  - Trying to use a variable that has not been declared
  - Missing headers generate complier errors
  - Misspelling or omitting ; , "" () will give you errors
- Logical errors may not cause errors
  - using = instead of ==
  - using + instead of -

Programming Cycle

- Write the source code
- Compile and Run
- Write the source code
- Compile and Run
- Fix bugs and rewrite
- Compile and Run
- Plus.....
- Fix bugs and rewrite
- Compile and Run

Questions