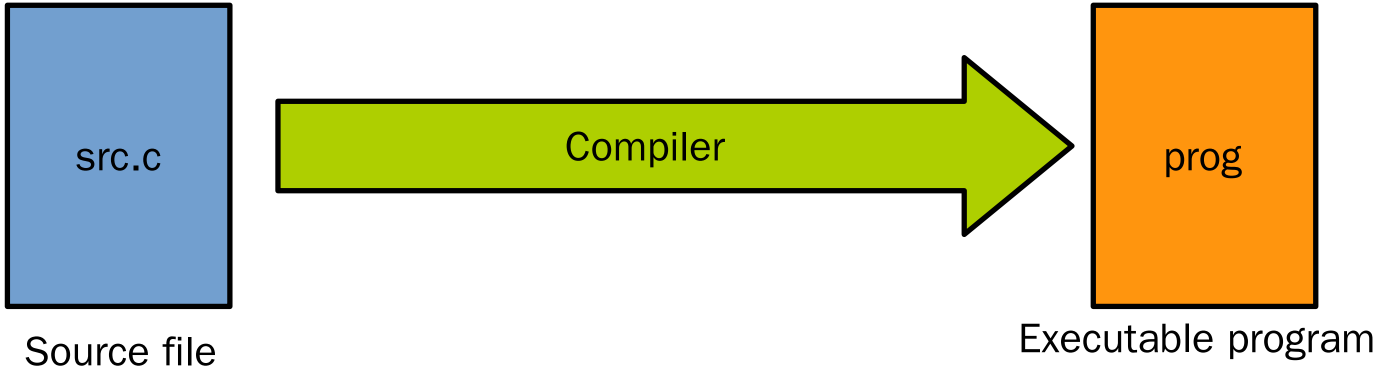
COMP26020 Programming Languages and Paradigms -- Part 1

The Preprocessor

The Preprocessor

```
gcc src.c -o prog
```



src.c

Compiler

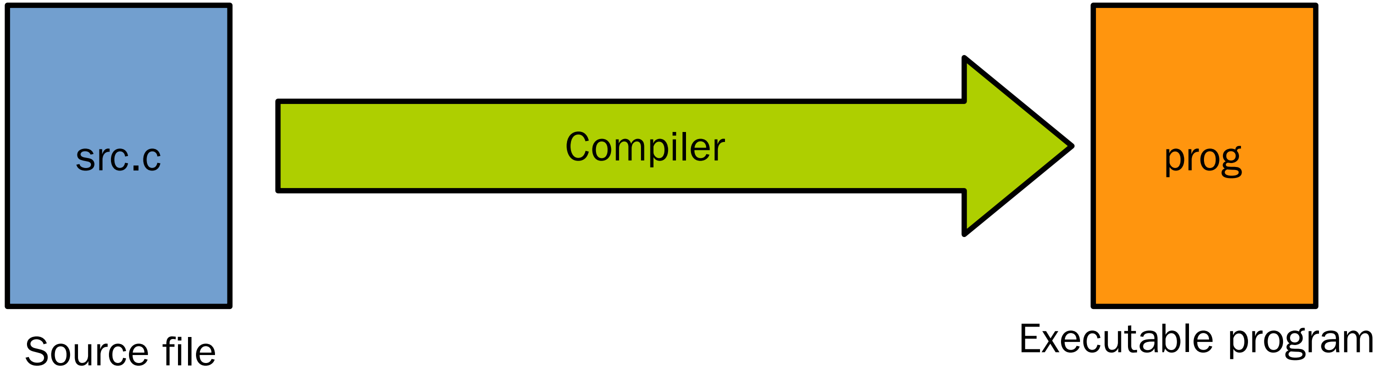
prog

Source file

Executable program

The Preprocessor

```
gcc src.c -o prog
```



src.c

Compiler

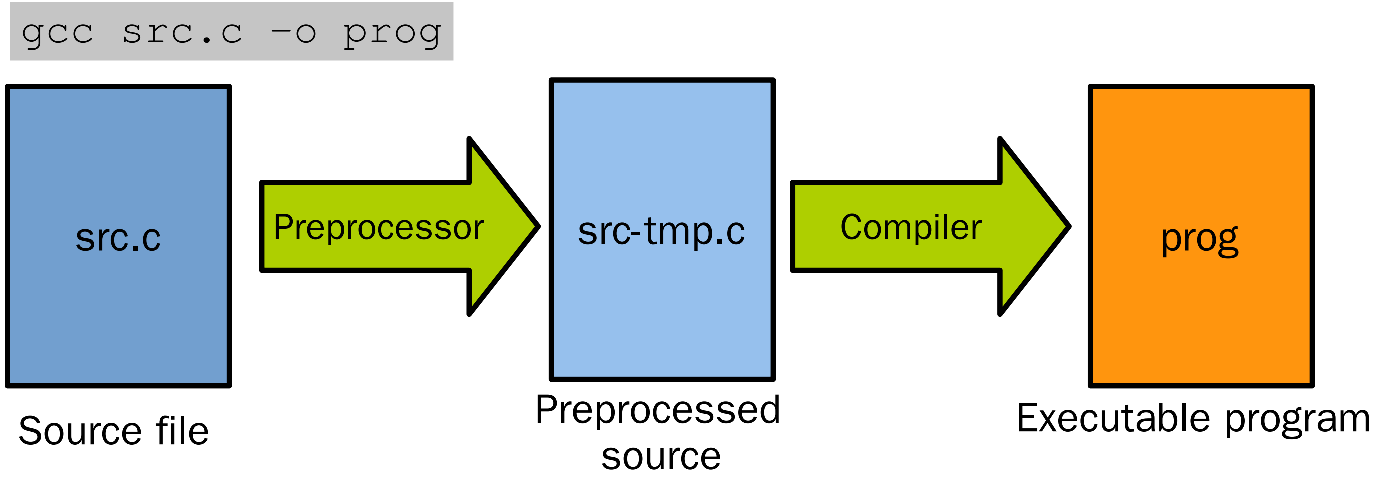
prog

Source file

Executable program

Actually things are a little bit more complicated 😛

The Preprocessor



```
gcc src.c -o prog
```

src.c

Source file

Preprocessor

src-tmp.c

Preprocessed source

Compiler

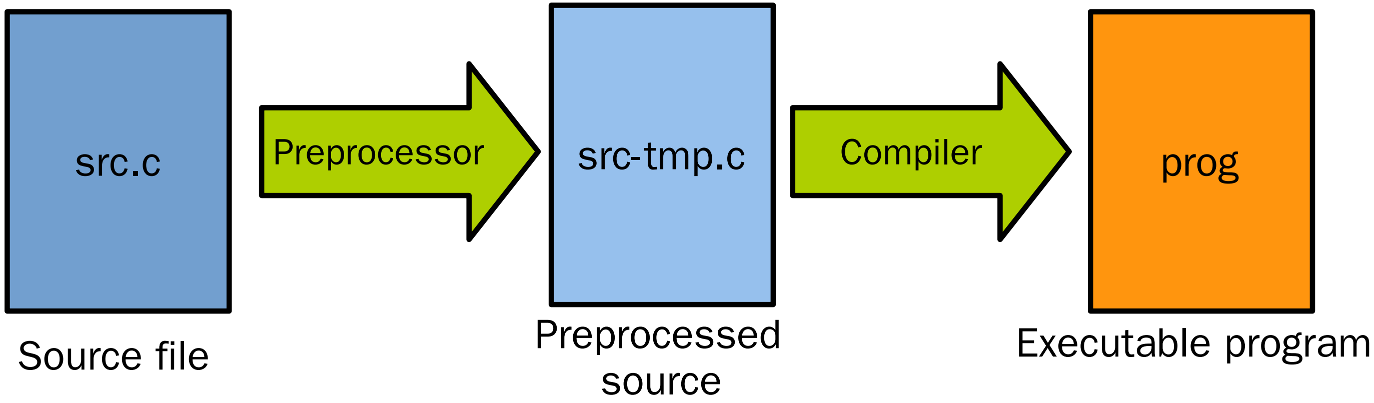
prog

Executable program

- Run by the compiler transparently
- Performs **textual transformations** in the source code:

The Preprocessor

```
gcc src.c -o prog
```



src.c

Source file

Preprocessor

src-tmp.c

Preprocessed source

Compiler

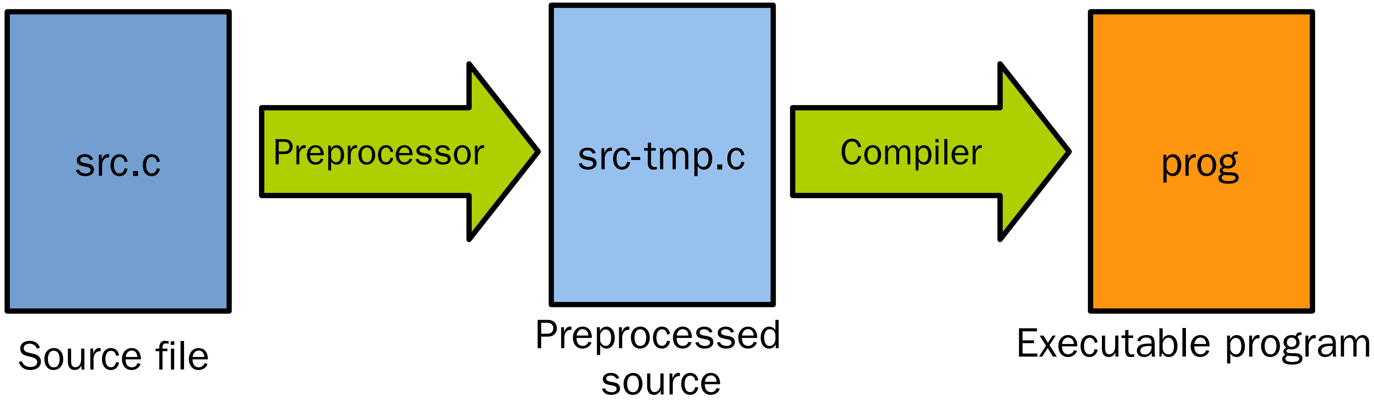
prog

Executable program

- Run by the compiler transparently
- Performs **textual transformations** in the source code:
    - Include **headers** to access functions, data structures, and other constructs defined in other source files

The Preprocessor

```
gcc src.c -o prog
```



src.c

Source file

Preprocessor

src-tmp.c

Preprocessed source

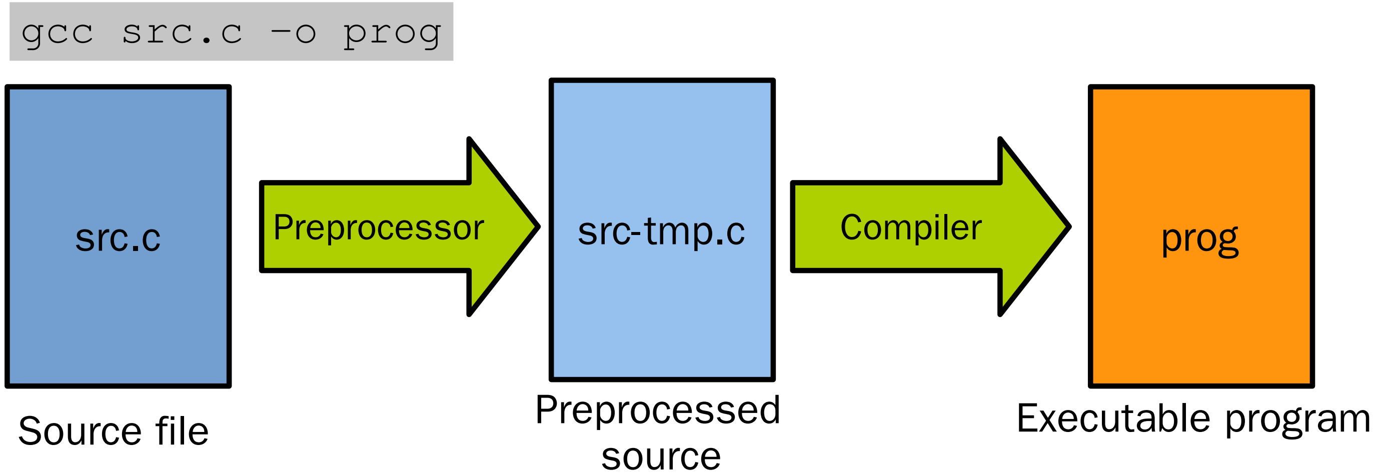
Compiler

prog

Executable program

- Run by the compiler transparently
- Performs **textual transformations** in the source code:
  - Include **headers** to access functions, data structures, and other constructs defined in other source files
  - Expand tokens named **macros** into more complex bits of code

The Preprocessor


`gcc src.c -o prog`

- Run by the compiler transparently
- Performs **textual transformations** in the source code:
  - Include **headers** to access functions, data structures, and other constructs defined in other source files
  - Expand tokens named **macros** into more complex bits of code
  - **Conditionally enable/disable some bits of code**

Header Inclusion

Header Inclusion

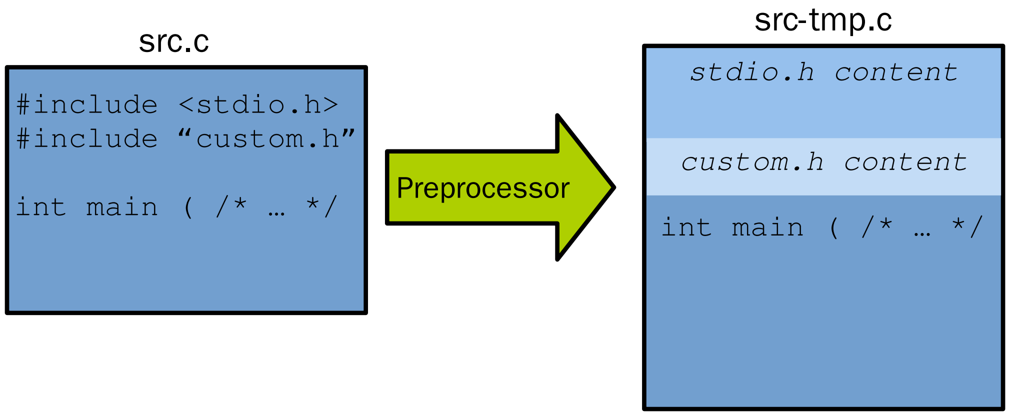
- **Header** files: C code file with `.h` extension, contain what is necessary to use foreign libraries/source files
    - Should be **included** in the source (`.c`) file wishing to use the foreign code

```c
// In the .c source file, include headers like that:
#include <stdio.h>            // Use <> to include a file from the default include path
#include "my-custom-header.h" // Use "" for the local and user-supplied include directories
```

Header Inclusion

- **Header** files: C code file with `.h` extension, contain what is necessary to use foreign libraries/source files
    - Should be **included** in the source (`.c`) file wishing to use the foreign code

```c
// In the .c source file, include headers like that:
#include <stdio.h>            // Use <> to include a file from the default include path
#include "my-custom-header.h" // Use "" for the local and user-supplied include directories
```



src.c

```
#include <stdio.h>
#include “custom.h”

int main ( /* … */
```

Preprocessor

src-tmp.c

stdio.h content

custom.h content

```
int main ( /* … */
```

Header Inclusion

- **Header** files: C code file with `.h` extension, contain what is necessary to use foreign libraries/source files
  - Should be **included** in the source (`.c`) file wishing to use the foreign code

```c
// In the .c source file, include headers like that:
#include <stdio.h>            // Use <> to include a file from the default include path
#include "my-custom-header.h" // Use "" for the local and user-supplied include directories
```

- Headers can be inclued in several `.c` source files composing a program
- To avoid double definitions, **they should contain only declarations:** functions prototypes, variables/custom types/structs declaration, etc.
- **No definitions** (i.e. function implementation or global variable affectation)

Macro Expansions

Macro Expansions

- Textual substitutions, useful for compile-time defined constants
    - Generally indicated in capital letters as a convention

Macro Expansions

- Textual substitutions, useful for compile-time defined constants
    - Generally indicated in capital letters as a convention

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int array[10];

  // If I modify the array size, I shall not
  // forget to update the iteration bound
  // here too!
  for(int i=0; i<10; i++) {
    array[i] = i;
    printf("array[%d] = %d\n", i, array[i]);
  }


  return 0;
}
```

Macro Expansions

- Textual substitutions, useful for compile-time defined constants
    - Generally indicated in capital letters as a convention

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int array[10];

  // If I modify the array size, I shall not
  // forget to update the iteration bound
  // here too!
  for(int i=0; i<10; i++) {
    array[i] = i;
    printf("array[%d] = %d\n", i, array[i]);
  }


  return 0;
}
```

```c
#include <stdio.h>

// If we want to change the array size,
// we only need to update this:
#define ARRAY_SIZE  10

int main(int argc, char **argv) {
  int array[ARRAY_SIZE];

  for(int i=0; i<ARRAY_SIZE; i++) {
    array[i] = i;
    printf("array[%d] = %d\n", i, array[i]);
  }

  return 0;
}
```

15-preprocessor/macro.c

Macro Expansions

- In general, **try to have a less hardcoded values as possible**
    - When you see that you are writing a constant value more then once, check if using a macro is possible!
    - Even if you have a constant used only once a macro can make it look more meaningful and easy to update in the future

Macro Expansions

- In general, **try to have a less hardcoded values as possible**
    - When you see that you are writing a constant value more then once, check if using a macro is possible!
    - Even if you have a constant used only once a macro can make it look more meaningful and easy to update in the future
- Macros can also be used to write macro "functions"
    - Can be error-prone, see here: https://bit.ly/3EaQ2DA

Macro Expansions

- Be careful with operator precedence! Use parentheses!

```c
#define SIZE_1  10
#define SIZE_2  10

// We can use a macro in another macro's
// definition:
#define TOTAL   SIZE_1 + SIZE_2

int main(int argc, char **argv) {

    // expect to print 10+10 = 20 * 2 = 40
    printf("total twice = %d\n", TOTAL * 2);

    return 0;
}
```

15-preprocessor/macro-replacement-issue.c

Macro Expansions

- Be careful with operator precedence! Use parentheses!

```c
#define SIZE_1  10
#define SIZE_2  10

// We can use a macro in another macro's
// definition:
#define TOTAL   SIZE_1 + SIZE_2

int main(int argc, char **argv) {

    // faulty! expands to: 10 + 10 * 2
    printf("total twice = %d\n", TOTAL * 2);

    return 0;
}
```

15-preprocessor/macro-replacement-issue.c

```c
#define SIZE_1  10
#define SIZE_2  10

// More than 1 macro in another macro's
// definition? use parentheses
#define TOTAL   (SIZE_1 + SIZE_2)

int main(int argc, char **argv) {

    // correct, expands to: (10 + 10) * 2
    printf("total twice = %d\n", TOTAL * 2);

    return 0;
}
```

15-preprocessor/macro-replacement-fix.c

Conditional Compilation

Conditional Compilation

```c
#define DEBUG_MODE             // controls the activation/deactivation of debug mode
#define VERBOSITY_LEVEL 4      // control the debug verbosity level

#ifdef DEBUG_MODE
int debug_function();
#endif

int main(int argc, char **argv) {
    printf("hello, world\n");

#ifdef DEBUG_MODE
    debug_function();
#endif
    return 0;
}

#ifdef DEBUG_MODE
int debug_function() {
    printf("This is printed only if the macro DEBUG_MODE is defined\n");

#if VERBOSITY_LEVEL > 3
    printf("Additional debug because the verbosity level is high\n");
#endif /* VERBOSITY_LEVEL */

    return 42;
}
#endif /* DEBUG_MODE */
```

[15-preprocessor/conditional-compilation.c]()

Conditional Compilation

```c
#define DEBUG_MODE          // comment this to disable debug mode
#define VERBOSITY_LEVEL 4

#ifdef DEBUG_MODE
int debug_function();
#endif

int main(int argc, char **argv) {
    printf("hello, world\n");

#ifdef DEBUG_MODE
    debug_function();
#endif
    return 0;
}

#ifdef DEBUG_MODE
int debug_function() {
    printf("This is printed only if the macro DEBUG_MODE is defined\n");

#if VERBOSITY_LEVEL > 3
    printf("Additional debug because the verbosity level is high\n");
#endif /* VERBOSITY_LEVEL */

    return 42;
}
#endif /* DEBUG_MODE */
```

15-preprocessor/conditional-compilation.c

Conditional Compilation

```c
#define DEBUG_MODE
#define VERBOSITY_LEVEL 4  // update this to set the level of verbosity

#ifdef DEBUG_MODE
int debug_function();
#endif

int main(int argc, char **argv) {
    printf("hello, world\n");

#ifdef DEBUG_MODE
    debug_function();
#endif
    return 0;
}

#ifdef DEBUG_MODE
int debug_function() {
    printf("This is printed only if the macro DEBUG_MODE is defined\n");

#if VERBOSITY_LEVEL > 3 // You can use the other C-like comparison operations (==, etc.)
    printf("Additional debug because the verbosity level is high\n");
#endif /* VERBOSITY_LEVEL */

    return 42;
}
#endif /* DEBUG_MODE */
```

15-preprocessor/conditional-compilation.c

Summary

- Preprocessor: textual transformation before compilation
- Automatically called by the compiler
- Header inclusion, macro expansion, conditional code inclusion

---

Feedback form: https://bit.ly/37s9JZ9