Hello.java (Page 1 of 1)

```java
//----------------------------------------------------------------------
// Hello.java
//----------------------------------------------------------------------

public class Hello
{
   // Write "hello, world\n" to the standard output stream.

   public static void main(String[] args)
   {
      System.out.println("hello, world");
   }
}
```

hello.c (Page 1 of 1)

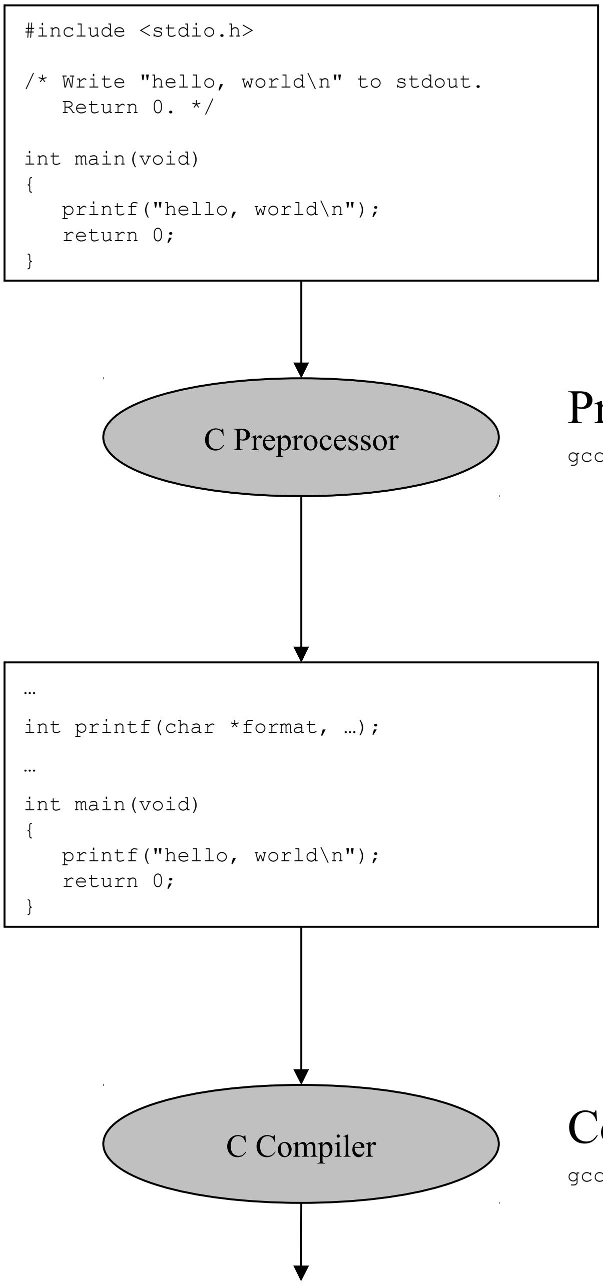
```c
/*--------------------------------------------------------------------*/
/* hello.c                                                            */
/*--------------------------------------------------------------------*/

#include <stdio.h>

/* Write "hello, world\n" to stdout. Return 0. */

int main(void)
{
   printf("hello, world\n");
   return 0;
}
```

Princeton University
COS 217: Introduction to Programming Systems
Building C Programs



```
#include <stdio.h>

/* Write "hello, world\n" to stdout.
   Return 0. */

int main(void)
{
   printf("hello, world\n");
   return 0;
}
```

hello.c
Source code
C language
Contains preprocessor directives

Preprocess

```
gcc217 -E hello.c > hello.i
```

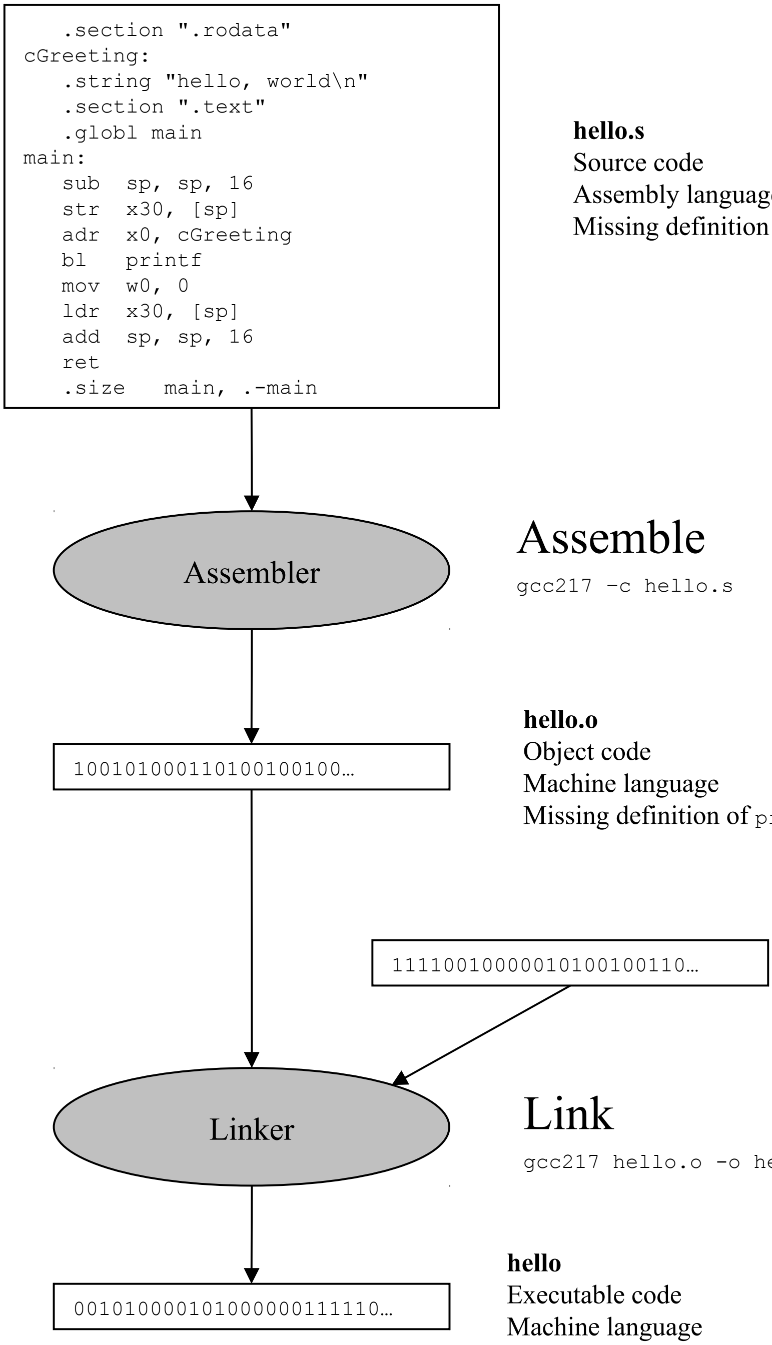
```
…
int printf(char *format, …);
…
int main(void)
{
   printf("hello, world\n");
   return 0;
}
```

hello.i
Source code
C language
Contains *declarations* of `printf()` and many other functions
Missing *definition* of `printf()`

Compile

```
gcc217 -S hello.i
```

Continued on next page



```
   .section ".rodata"
cGreeting:
   .string "hello, world\n"
   .section ".text"
   .globl main
main:
   sub  sp, sp, 16
   str  x30, [sp]
   adr  x0, cGreeting
   bl   printf
   mov  w0, 0
   ldr  x30, [sp]
   add  sp, sp, 16
   ret
   .size   main, .-main
```

hello.s
Source code
Assembly language
Missing definition of `printf()`

Assemble

`gcc217 -c hello.s`

hello.o
Object code
Machine language
Missing definition of `printf()`

libc.a
Library containing
machine language definitions
of `printf()` and many
other functions

Link

`gcc217 hello.o -o hello`

hello
Executable code
Machine language
Contains definition of `printf()`

Shortcut:

`gcc217 hello.c -o hello`

`gcc217`

is an abbreviation for

```
gcc -Wall -Wextra -Wno-unused-parameter
   -ansi -pedantic
```

circle1.c (Page 1 of 1)

```
/*---------------------------------------------------------------------*/
/* circle1.c                                                           */
/* Author: Bob Dondero                                                 */
/*---------------------------------------------------------------------*/

#include <stdio.h>

/*---------------------------------------------------------------------*/

/* Read a circle’s radius from stdin, and compute and write its
   diameter and circumference to stdout.  Return 0. */

int main(void)
{
   int iRadius;
   int iDiam;
   double dCircum;

   printf("Enter the circle’s radius:\n");
   scanf("%d", &iRadius);

   iDiam = 2 * iRadius;
   dCircum = 3.14159 * (double)iDiam;

   printf("A circle with radius %d has diameter %d\n",
          iRadius, iDiam);
   printf("and circumference %f.\n", dCircum);
   return 0;
}

/*---------------------------------------------------------------------*/

/* Sample executions:

$ gcc217 circle1.c -o circle1

$ ./circle1
Enter the circle’s radius:
5
A circle with radius 5 has diameter 10
and circumference 31.415900.

$ ./circle1
Enter the circle’s radius:
1
A circle with radius 1 has diameter 2
and circumference 6.283180.

*/
```

circle2.c (Page 1 of 1)

```
/*--------------------------------------------------------------------*/
/* circle2.c                                                      */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>

/*--------------------------------------------------------------------*/

/* Read a circle's radius from stdin, and compute and write its
   diameter and circumference to stdout.  Return 0 if successful. */

int main(void)
{
   int iRadius;
   int iDiam;
   double dCircum;
   const double PI = 3.14159;  /* or (4.0 * atan(1.0)) */

   printf("Enter the circle's radius:\n");
   if (scanf("%d", &iRadius) != 1)
   {
      fprintf(stderr, "Error: Not a number\n");
      exit(EXIT_FAILURE);
   }

   iDiam = 2 * iRadius;
   dCircum = PI * (double)iDiam;

   printf("A circle with radius %d has diameter %d\n",
          iRadius, iDiam);
   printf("and circumference %f.\n", dCircum);
   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample executions:

$ gcc217 circle2.c -o circle2

$ ./circle2
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.415900.

$ ./circle2
Enter the circle's radius:
1
A circle with radius 1 has diameter 2
and circumference 6.283180.

$ ./circle2
Enter the circle's radius:
abc
Error: Not a number

*/
```

Princeton University
COS 217: Introduction to Programming Systems
C Symbolic Constants

Approach 1: Macros

Example

```
int main(void)
{
   #define START_STATE 0
   #define POSSIBLE_COMMENT_STATE 1
   #define COMMENT_STATE 2
   ...
   int iState;
   ...
   iState = START_STATE;
   ...
}
```

Terminology

`START_STATE`, `POSSIBLE_COMMENT_STATE`, and `COMMENT_STATE` are *macros*.

Strengths

Preprocessor does substitutions only for tokens.

```
int iSTART_STATE;  /* No substitution. */
```

Preprocessor does not do substitutions within string literals.

```
printf("What is the START_STATE?\n"); /* No substitution. */
```

Simple textual substitution; works for any type of data.

```
#define PI 3.14159
```

Weaknesses

Preprocessor does not respect context.

```
int START_STATE;
```

After preprocessing, becomes:

```
int 0;  /* Compiletime error. */
```

Convention: Use all uppercase letters to reduce probability of unintended replacement.

Preprocessor does not respect scope.

Preprocessor replaces `START_STATE` with 0 from point of `#define` to end of *file*, not to end of *function*. Could affect subsequent functions unintentionally.

Convention: Place `#define`s at beginning of file, not within function definitions

Approach 2: Constant Variables

Example

```
int main(void)
{
   const int START_STATE = 0;
   const int POSSIBLE_COMMENT_STATE = 1;
   const int COMMENT_STATE = 2;
   ...
   ...
   int iState;
   ...
   iState = START_STATE;
   ...
   iState = COMMENT_STATE;
   ...
}
```

Strengths

Works for any type of data.

```
const double PI = 3.14159;
const long MAX = 1000000000000000000L;
```

Handled by compiler; compiler respects context and scope.

Weaknesses

Does not work for array lengths (unlike C99, C11, and C++).

```
const int ARRAY_LENGTH = 10;
...
int aiNumbers[ARRAY_LENGTH];  /* Compile-time warning */
...
```

Approach 3: Enumerations

Example

```
int main(void)
{
   enum State {START_STATE, POSSIBLE_COMMENT_STATE, COMMENT_STATE, ...};
   enum State eState;
   ...
   eState = START_STATE;
   ...
   eState = COMMENT_STATE;
   ...
}
```

Terminology

`enum State` is an *enumeration type.*
`START_STATE, POSSIBLE_COMMENT_STATE`, … are *enumeration constants*.
`eState` is an *enumeration*; it is of type `enum State.`

Notes

Can use an expression of type `int` where an enumeration constant is expected.

```
eState = 0;      /* Can assign an int to an enumeration. */
```

Can use an enumeration constant where an expression of type `int` is expected.

```
i = START_STATE;  /* Can assign an enumeration constant to an int variable.
                     START_STATE is an alias for 0, POSSIBLE_COMMENT_STATE
                     is an alias for 1, etc. */
```

Strengths

Can explicitly specify values for enumeration constants.

```
enum State {START_STATE=5, POSSIBLE_COMMENT_STATE=3, COMMENT_STATE=4, ...};
```

Can define an *anonymous* enumeration type, thus effectively giving symbolic names to `int` literals.

```
enum {MAX_VALUE = 9999};
...
int i;
...
i = MAX_VALUE;
...
```

Works when specifying array lengths.

```
enum {ARRAY_LENGTH = 10};
...
int aiNumbers[ARRAY_LENGTH];
...
```

Weakness

Works only for `int` literals.

```
enum {PI = 3.14159};                  /* Compile-time error */
enum {MAX = 10000000000000000000L};    /* Compile-time warning */
```

Style Rules

To give a symbolic name to a literal of type ...	Use …
`int`	An enumeration

`char`
`unsigned char`
`short`
`unsigned short`
`unsigned int`
`long`
`unsigned long`
`float`
`double`
`long double`
string
	A constant variable

Don't use macros to give symbolic names to literals.