C

```
#include <stdio.h> int main(){printf("Hello, World!\n");return 0;}
```

C++

```
#include <iostream>\nint main(){std::cout << "Hello, World!" << std::endl;}
```

C#

```
using System;\nclass Hello{static void Main(){Console.WriteLine("Hello,
World!");}}
```

Java

```
public class HelloWorld{public static void main(String[]
args){System.out.println("Hello, World!");}}
```

Python

```
print("Hello, World!")
```

Ruby

```
puts "Hello, World!"
```

JavaScript

```
console.log("Hello, World!");
```

PHP

```
&lt;?php echo "Hello, World!"; ?&gt;
```

Bash

```
echo "Hello, World!"
```

PowerShell

```
Write-Output "Hello, World!"
```

Go

```
package main\nimport "fmt"\nfunc main(){fmt.Println("Hello, World!")}
```

Rust

```
fn main(){println!("Hello, World!");}
```

Swift

```
print("Hello, World!")
```

Kotlin

```
fun main(){println("Hello, World!")}
```

Perl

```
print "Hello, World!\n";
```

Lua

```
print("Hello, World!")
```

Haskell

```
main = putStrLn "Hello, World!"
```

R

```
cat("Hello, World!\n")
```

MATLAB

```
disp('Hello, World!')
```

Scala

```
object HelloWorld extends App {println("Hello, World!")}
```

Dart

```
void main(){print('Hello, World!');}
```

TypeScript

```
console.log('Hello, World!');
```

Elixir

```
IO.puts "Hello, World!"
```

Julia

```
println("Hello, World!")
```

F#

```
printfn "Hello, World!"
```

Objective-C

```
#import &lt;Foundation/Foundation.h&gt;\nint
main(){@autoreleasepool{NSLog(@"Hello, World!");}}
```

Visual Basic .NET

```
Module Hello\n Sub Main()\n Console.WriteLine("Hello, World!")\n End
Sub\nEnd Module
```

Pascal

```
program HelloWorld;begin writeln('Hello, World!');end.
```

Fortran

```
program hello\n print *, "Hello, World!"\nend program hello
```

COBOL

```
IDENTIFICATION DIVISION.\nPROGRAM-ID. HELLO.\nPROCEDURE DIVISION.\nDISPLAY
"Hello, World!".\nSTOP RUN.
```

Ada

```
with Ada.Text_IO; use Ada.Text_IO;\nprocedure Hello is begin\n Put_Line
("Hello, World!");\nend Hello;
```

Assembly (NASM, x86-Linux)

```
section .data\n msg db "Hello, World!",10\n len equ $-msg\nsection .text\n
global _start\n_start:\n mov eax,4 ; sys_write\n mov ebx,1 ; stdout\n mov
ecx,msg\n mov edx,len\n int 0x80\n mov eax,1 ; sys_exit\n xor ebx,ebx\n int
0x80
```

Brainfuck

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-
]>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>.
```

Clojure

```
(println "Hello, World!")
```

Lisp (Common)

```
(format t "Hello, World!~%")
```

Scheme

```
(display "Hello, World!") (newline)
```

Prolog

```
:- initialization(main).\nmain :- write('Hello, World!'), nl, halt.
```

Erlang

```
-module(hello).\n-export([start/0]).\nstart() -> io:format("Hello,
World!~n").
```

OCaml

```
print_endline "Hello, World!";;
```

Haxe

```
class Main {static function main(){trace("Hello, World!");}}
```

Smalltalk

```
Transcript show: 'Hello, World!'; cr.
```

Tcl

```
puts "Hello, World!"
```

Racket

```
#lang racket\n(displayln "Hello, World!")
```

Groovy

```
println 'Hello, World!'
```

LuaJIT

```
print("Hello, World!")
```

Crystal

```
puts "Hello, World!"
```

Nim

```
echo "Hello, World!"
```

D

```
import std.stdio;\nvoid main(){writeln("Hello, World!");}
```

Dart (Flutter)

```
void main(){runApp(const Center(child: Text('Hello, World!')));}
```

Forth

```
." Hello, World!" CR
```

PL/SQL

```
BEGIN DBMS_OUTPUT.PUT_LINE('Hello, World!'); END;
```

SQL (MySQL)

```
SELECT 'Hello, World!' AS greeting;
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL; entity hello is end hello;
architecture behav of hello is begin process begin report "Hello, World!";
wait; end process; end behav;
```

```
module hello; initial $display("Hello, World!"); endmodule
```

LaTeX (document)

```
\documentclass{article}\begin{document}Hello, World!\end{document}
```

Markdown (code-fence)

```
1
text\nHello, World!\n
```

| | Racket |

```
(displayln "Hello, World!")
```

| | AWK |

```
BEGIN {print "Hello, World!"}
```

| | Sed |

```
s/.*/Hello, World!/
```

| | Makefile |

```
all:\n\t@echo "Hello, World!"
```

| | Dockerfile |

```
FROM alpine\nCMD ["echo","Hello, World!"]
```

| | YAML (anchor) |

```
message: "Hello, World!"
```

| | JSON |

```
{ "message": "Hello, World!" }
```

| | XML |

```
<message>Hello, World!</message>
```

| | HTML |

```
<!DOCTYPE html>\n<html><body>Hello, World!</body></html>
```

| | CSS |

```
body::after {content: "Hello, World!";}
```

| | Sass |

```
body:before\n content: "Hello, World!"
```

| | Less |

```
body:before { content: "Hello, World!"; }
```

| | CoffeeScript |

```
console.log "Hello, World!"
```

| | TypeScript (Deno) |

```
console.log("Hello, World!");
```

| | Zig |

```
const std = @import("std");\npub fn main() void { std.debug.print("Hello,
World!\n", .{}); }
```

| | Crystal |

```
puts "Hello, World!"
```

| | Haskell (GHCi) |

```
putStrLn "Hello, World!"
```

| | Idris |

```
module Main\nmain : IO ()\nmain = putStrLn "Hello, World!"
```

| | J |

```
'Hello, World!'
```

| | Kotlin/Native |

```
fun main() = println("Hello, World!")
```

| | Lisp (ClojureScript) |

```
(.log js/console "Hello, World!")
```

| | MATLAB (Octave) |

```
disp('Hello, World!')
```

| | Mercury |

```
:- module hello.\n:- interface.\n:- pred main(io::di, io::uo) is det.\n:-
implementation.\nmain(!IO) :- io.write_string("Hello, World!\n", !IO).
```

| | Nix |

```
let msg = "Hello, World!"; in builtins.trace msg null
```

| | OCaml (utop) |

```
print_endline "Hello, World!";;
```

| | Perl6 (Raku) |

```
say "Hello, World!";
```

| | PostScript |

```
(Hello, World!) show
```

| | Prolog (SWI) |

```
:- initialization(main).\nmain :- writeln('Hello, World!'), halt.
```

| | PureScript |

```
main = log "Hello, World!"
```

| | Racket (typed) |

```
#lang typed/racket\n(: main (-> Void))\n(define (main) (displayln "Hello,
World!"))
```

| | REBOL |

```
print "Hello, World!"
```

| | RPG |

```
**FREE\nDcl-S msg VarChar(25) Inz('Hello, World!');\nDsply msg;
```

| | Rust (edition 2021) |

```
fn main(){println!("Hello, World!");}
```

| | Scala (2) |

```
object Hello extends App {println("Hello, World!")}
```

| | Scheme (Racket) |

```
#lang racket\n(displayln "Hello, World!")
```

| | Smalltalk (Pharo) |

```
Transcript show: 'Hello, World!'; cr.
```

| | Solidity |

```
pragma solidity ^0.8.0;\ncontract HelloWorld {function hello() public pure
returns (string memory){return "Hello, World!";}}
```

| | SPARK |

```
with Ada.Text_IO; use Ada.Text_IO;\nprocedure Hello is begin Put_Line
("Hello, World!"); end Hello;
```

| | Starlark |

```
print("Hello, World!")
```

| | Swift (Linux) |

```
print("Hello, World!")
```

| | TLA+ |

```
------------------------------ MODULE Hello ------------------------------
\nEXTENDS Naturals\nTHEOREM Hello == ASSUME TRUE PROVE \n (\E x \in Nat : x
= 0) * dummy proof — in TLA+ every spec is a
program\n==========================================================================
==========
```

| | Vala |

```
void main(){print("Hello, World!\n");}
```

| | Verilog-AMS |

```
timescale 1ns/1ps\nmodule hello; initial $display("Hello, World!");
endmodule
```

| | VHDL (2008) |

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL; entity hello is end; architecture
sim of hello is begin process begin report "Hello, World!"; wait; end
process; end;
```

| | XSLT |

```
&lt;xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"&gt;\n &lt;xsl:template
match="/"&gt;\n &lt;output&gt;Hello, World!&lt;/output&gt;\n
&lt;/xsl:template&gt;\n&lt;/xsl:stylesheet&gt;
```

| | Yacc |

```
%{\n#include &lt;stdio.h&gt;\n%}\n%%\nprogram: /* empty */\n | program '\\n'
{\n printf("Hello, World!\\n");\n }\n%%\nint main(){yyparse();}
```

| | Zsh |

```
echo "Hello, World!"
```

| | Ada |

```
with Ada.Text_IO; use Ada.Text_IO;\nprocedure Hello is begin Put_Line
("Hello, World!"); end Hello;
```

| | APL |

```
 Hello, World!\n'Hello, World!'
```

| | AppleScript |

```
display dialog "Hello, World!"
```

| | Arduino (C++) |

```
void setup(){Serial.begin(9600); Serial.println("Hello, World!");}\nvoid
loop(){}
```

| | B |

```
main(){printf("Hello, World!\\n");}
```

| | BASIC (QBASIC) |

```
PRINT "Hello, World!"
```

| | Bash (POSIX) |

```
printf "Hello, World!\n"
```

| | Befunge |

```
"Hello, World!"&gt;:#,_@
```

| | Boo |

```
print "Hello, World!"
```

| | C (Objective-C++) |

```
#import &lt;Foundation/Foundation.h&gt;\nint main(){NSLog(@"Hello, World!");
return 0;}
```

| | Chapel |

```
writeln("Hello, World!");
```

| | ChucK |

```
println("Hello, World!");
```

| | Clipper |

```
? "Hello, World!"
```

| | CobolScript |

```
IDENTIFICATION DIVISION.\nPROGRAM-ID. Hello.\nPROCEDURE DIVISION.\nDISPLAY "Hello, World!".\nSTOP RUN.
```

| | ColdFusion |

```
<cfoutput>Hello, World!</cfoutput>
```

| | Common Lisp |

```
(format t "Hello, World!~%")
```

| | D (DMD) |

```
import std.stdio;\nvoid main(){writeln("Hello, World!");}
```

| | Dart (CLI) |

```
void main(){print('Hello, World!');}
```

| | Delphi |

```
program HelloWorld;\nbegin\n WriteLn('Hello, World!');\nend.
```

| | Dylan |

```
format-out("Hello, World!%n");
```

| | Eiffel |

```
class HELLO\ncreate make\nfeature\n make do\n print ("Hello, World!%N")\n end\nend
```

| | Elixir (Mix) |

```
IO.puts "Hello, World!"
```

| | Elm |

```
module Main exposing (main)\nimport Browser\nmain = Browser.sandbox { init = (), view = \\_ -> Html.text "Hello, World!", update = \\_ _ -> () }
```

| | Emacs Lisp |

```
(message "Hello, World!")
```

| | Erlang (OTP) |

```
-module(hello).\n-export([hello/0]).\nhello() -> io:format("Hello,
World!~n").
```

| | F# (script) |

```
printfn "Hello, World!"
```

| | Factor |

```
"Hello, World!" print
```

| | Falcon |

```
print("Hello, World!")
```

| | FAUST |

```
process = _:+("Hello, World!");
```

| | Forth (gforth) |

```
." Hello, World!" CR
```

| | GDScript (Godot) |

```
func _ready():\n print("Hello, World!")
```

| | Golo |

```
module hello\nfunction main = |args| -> println("Hello, World!")
```

| | Gosu |

```
println("Hello, World!")
```

| | Groovy (script) |

```
println 'Hello, World!'
```

| | Hack |

```
<?hh\n<<__EntryPoint>>\nfunction main(): void { echo "Hello, World!\n"; }
```

| | Haskell (GHCi) |

```
putStrLn "Hello, World!"
```

| | Haxe (hx) |

```
class Hello { static function main() { trace("Hello, World!"); } }
```

| | IDL (Interface Definition Language) |

```
interface Hello { void sayHello(); };
```

| | Io |

```
"Hello, World!" println
```

| | J (JAva) |

```
'Hello, World!'
```

| | Jolie |

```
define HelloWorld { println("Hello, World!"); }
```

| | Julia (script) |

```
println("Hello, World!")
```

| | K (Kona) |

```
"Hello, World!"
```

| | Kotlin (JVM) |

```
fun main() { println("Hello, World!") }
```

| | LabVIEW (G) |

```
[Hello World] (text constant)
```

| | Lisp (Scheme) |

```
(display "Hello, World!") (newline)
```

| | LiveCode |

```
put "Hello, World!" into field "output"
```

| | Logtalk |

```
:- object(hello).\n :- public([say/0]).\n say :- write('Hello, World!'),
nl.\n:- end_object.
```

| | LOLCODE |

```
HAI 1.2\n VISIBLE "Hello, World!"\nKTHXBYE
```

| | LotusScript |

```
Msgbox "Hello, World!"
```

| | Magik |

```
_print "Hello, World!"
```

| | Mathematica |

```
Print["Hello, World!"]
```

| | M4 |

```
define(msg', `Hello, World!')msg
```

| | Max/MSP (JavaScript) |

```
post("Hello, World!\n");
```

| | Mercury (imperative) |

```
:- module hello.\n:- interface.\n:- pred main(io::di, io::uo) is det.\n:-
implementation.\nmain(!IO) :- io.write_string("Hello, World!\n", !IO).
```

| | Mesa |

```
MODULE Hello;\nBEGIN\n Out.Text := "Hello, World!";\nEND Hello.
```

| | MiniScript |

```
print "Hello, World!"
```

| | MQL4 |

```
void OnStart(){Print("Hello, World!");}
```

| | MUMPS |

```
WRITE "Hello, World!",!
```

| | NATURAL |

```
WRITE 'Hello, World!'
```

| | Nemerle |

```
System.Console.WriteLine("Hello, World!");
```

| | NetLogo |

```
observer> print "Hello, World!"
```

| | Nimrod |

```
echo "Hello, World!"
```

| | OPL (IBM ILOG) |

```
execute { writeln("Hello, World!"); }
```

| | OpenSCAD |

```
echo("Hello, World!");
```

| | Oz |

```
{Browse "Hello, World!"}
```

| | PARI/GP |

```
print("Hello, World!\n")
```

| | Pascal (FreePascal) |

```
program Hello; begin writeln('Hello, World!'); end.
```

| | Pikespeak |

```
print "Hello, World!"
```

| | PL/I |

```
PUT EDIT('Hello, World!');
```

| | PL/SQL (Oracle) |

```
BEGIN DBMS_OUTPUT.PUT_LINE('Hello, World!'); END;
```

| | PostScript (Level 2) |

```
(Hello, World!) =
```

| | PowerBuilder |

```
MessageBox("Hello", "World!")
```

| | Prolog (SWI) |

```
:- initialization(main).\nmain :- writeln('Hello, World!'), halt.
```

| | PureBasic |

```
MessageRequester("Hello","World")
```

| | Q# |

```
operation SayHello() : Unit { Message("Hello, World!"); }
```

| | R (Rscript) |

```
cat("Hello, World!\n")
```

| | Racket (Typed) |

```
#lang typed/racket\n(: hello (-> Void))\n(define (hello) (printf "Hello,
World!\n"))\n(hello)
```

| | REBOL 2 |

```
print "Hello, World!"
```

| | Red |

```
print "Hello, World!"
```

| | ReScript |

```
Js.log("Hello, World!")
```

| | Rexx |

```
say 'Hello, World!'
```

| | RPGLE |

```
*FREE\ndsply 'Hello, World!'
```

| | Ruby (irb) |

```
puts "Hello, World!"
```

| | Rust (stable) |

```
fn main() { println!("Hello, World!"); }
```

| | SAS |

```
%put Hello, World!;
```

| | Scala (3) |

```
@main def hello = println("Hello, World!")
```

| | Scheme (R5RS) |

```
(display "Hello, World!") (newline)
```

| | Scratch |

```
[when green flag clicked] → say "Hello, World!" for 2 secs
```

| | Simula |

```
Begin\n OutText("Hello, World!");\n End.
```

| | Smalltalk (Pharo) |

```
Transcript show: 'Hello, World!'; cr.
```

| | Solidity (0.8) |

```
pragma solidity ^0.8.0;\ncontract Hello { function greet() public pure
returns (string memory){ return "Hello, World!"; } }
```

| | SPARK (Ada) |

```
with Ada.Text_IO; use Ada.Text_IO; procedure Hello is begin Put_Line
("Hello, World!"); end Hello;
```

| | Squirrel |

```
print("Hello, World!");
```

| | Stata |

```
display "Hello, World!"
```

| | Swift (macOS) |

```
print("Hello, World!")
```

| | Tcl (expect) |

```
puts "Hello, World!"
```

| | TypeScript (node) |

```
console.log('Hello, World!');
```

| | UnityScript |

```
print("Hello, World!");
```

| | V (Vale) |

```
fn main() { println('Hello, World!') }
```

| | Vala (GNOME) |

```
void main(){print("Hello, World!\n");}
```

| | Verilog-AMS |

```
module hello; initial $display("Hello, World!"); endmodule
```

| | Visual Basic 6 |

```
MsgBox "Hello, World!"
```

| | Visual Basic .NET |

```
Module Hello\n Sub Main()\n Console.WriteLine("Hello, World!")\n End
Sub\nEnd Module
```

| | WLang |

```
out "Hello, World!"
```

| | X10 |

```
public class HelloWorld { public static def main(Rail[String] args) {
Console.OUT.println("Hello, World!"); } }
```

| | Xojo |

```
MessageBox("Hello, World!")
```

| | Yacc (Bison) |

```
%{\n#include &lt;stdio.h&gt;\n%}\n%%\nprogram: / empty */\n | program '\n' {
printf("Hello, World!\n"); }\n%%\nint main(){yyparse();}
```

| | Zig |

```
const std = @import("std");\npub fn main() void { std.debug.print("Hello, World!\n", .{}); }
```

| | Zsh (shell) |

```
echo "Hello, World!"
```