



# Computer Languages and Lab Preliminaries

---

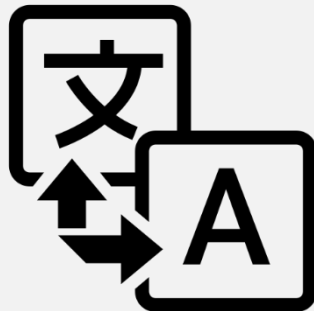
**Prof. Joongheon Kim**  
**Korea University, School of Electrical Engineering**  
<https://joongheon.github.io>  
[joongheon@korea.ac.kr](mailto:joongheon@korea.ac.kr)



- **Compile and Execution**
- Overview of C



## Natural Languages



Korean,  
English,  
Chinese,  
French, ...

## Formal Languages

A screenshot of a code editor showing JavaScript code. The code includes a function 'revItem' that takes 'portPostId' as an argument and returns 'portPrev'. It also shows a 'revItem' function call and a 'revItem' function definition. The code is color-coded and has a dark background.

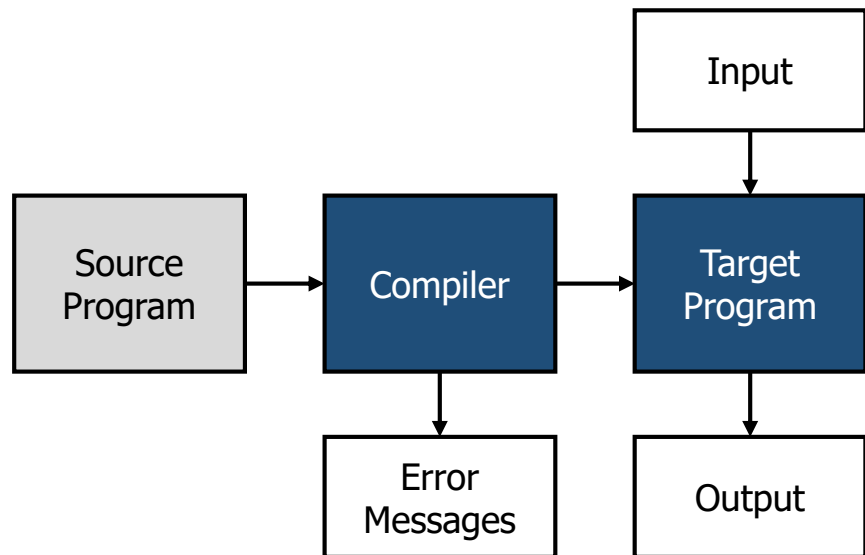
C, C++,  
Java,  
Python,  
JavaScript, ...



# Compilers and Interpreters

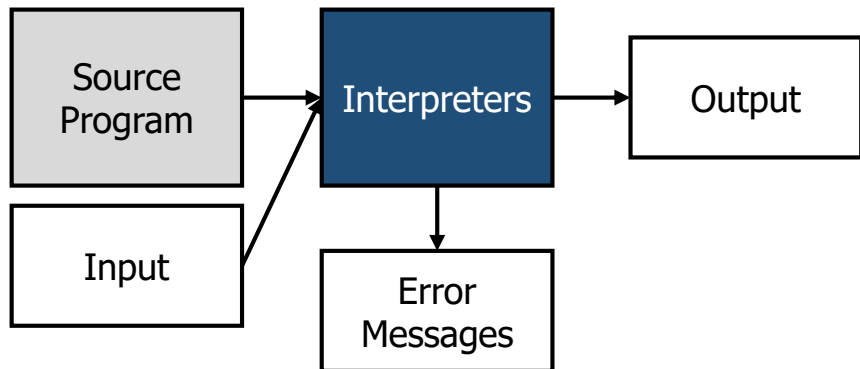
## Compilation

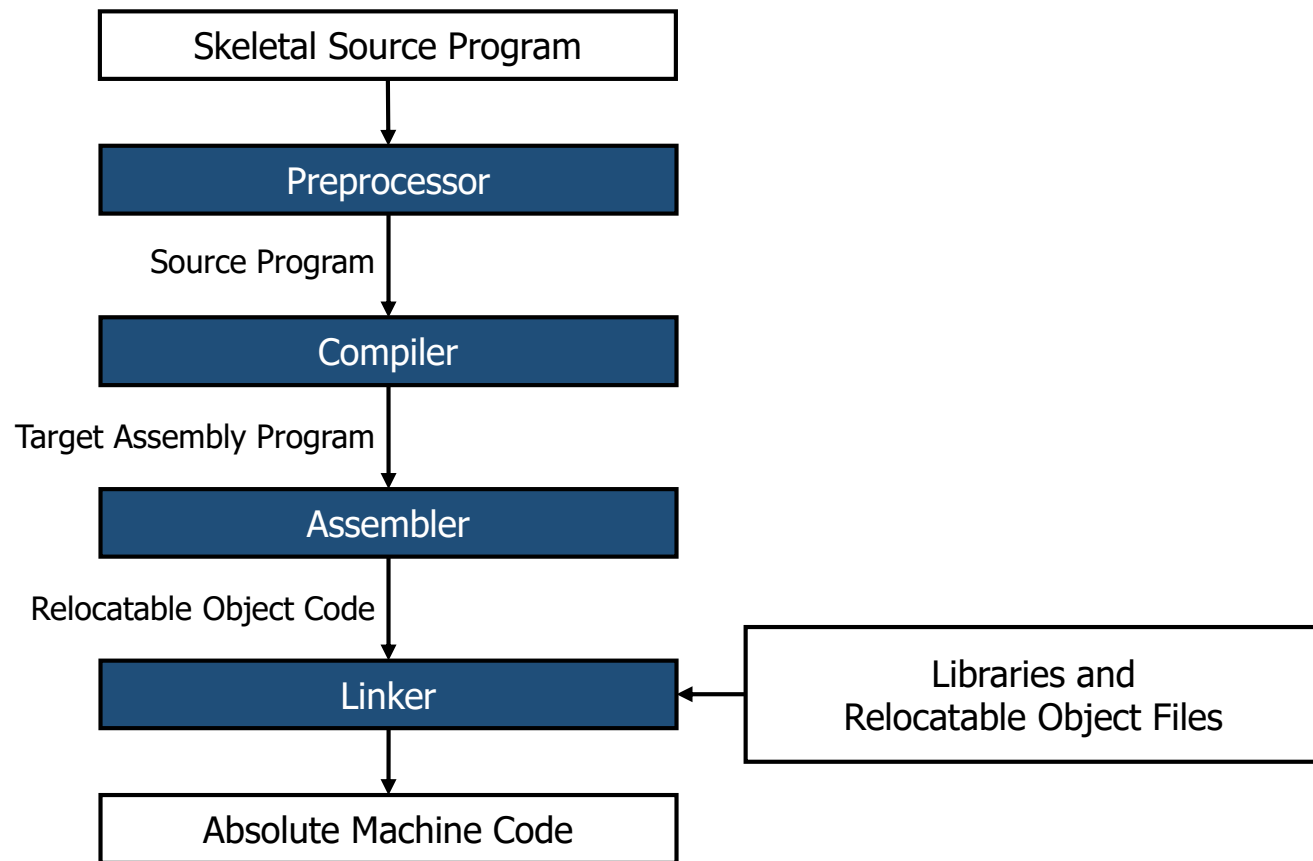
Translation of a program written in a source language into a semantically equivalent program written in a target language (e.g., C, Java, ...).



## Interpretation

Performing the operations implied by the source program (e.g., matlab, python, ...).







# The Phases of a Compiler

Phase	Output	Sample
<i>Programmer (source code producer)</i>	Source string	<b>A=B+C ;</b>
<i>Scanner (performs lexical analysis)</i>	Token string	<b>'A', '=', 'B', '+', 'C', ';'</b> And <i>symbol table</i> with names
<i>Parser (performs syntax analysis based on the grammar of the programming language)</i>	Parse tree or abstract syntax tree	<pre>       ;               =     /  \    A    +       /  \      B    C </pre>
<i>Semantic analyzer (type checking, etc)</i>	Annotated parse tree or abstract syntax tree	
<i>Intermediate code generator</i>	Three-address code, quads, or RTL	<pre> <b>int2fp</b> B      t1 +      t1      C      t2 :=      t2      A </pre>
<i>Optimizer</i>	Three-address code, quads, or RTL	<pre> <b>int2fp</b> B      t1 +      t1      #2.3 A </pre>
<i>Code generator</i>	Assembly code	<pre> <b>MOVF</b>  #2.3,r1 <b>ADDF2</b> r1,r2 <b>MOVF</b>  r2,A </pre>
<i>Peephole optimizer</i>	Assembly code	<pre> <b>ADDF2</b> #2.3,r2 <b>MOVF</b>  r2,A </pre>



- Execution without Compilation (C Language)
  - [http://www.tutorialspoint.com/compile\\_c\\_online.php](http://www.tutorialspoint.com/compile_c_online.php)

```
codingground | Compile and Execute C Online (GNU GCC v7.1.1)
SIMPLY EASY CODING

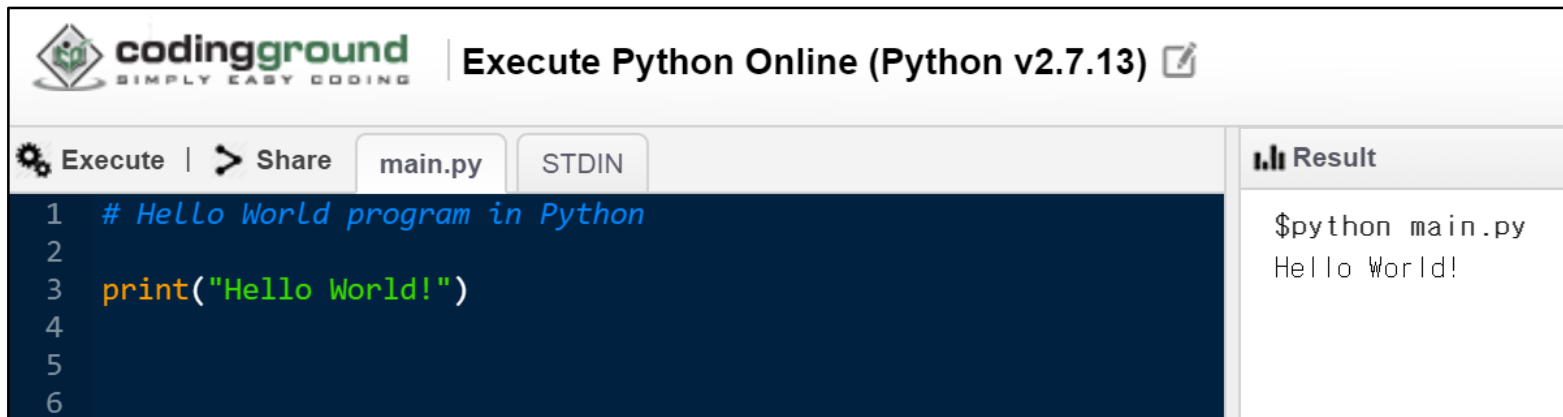
Execute | Share | main.c | STDIN | Result

1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello, World!\n");
6
7      return 0;
8  }
9

$gcc -o main *.c
$main
Hello, World!
```



- Execution without Compilation (Python Language)
  - [https://www.tutorialspoint.com/execute\\_python\\_online.php](https://www.tutorialspoint.com/execute_python_online.php)



The screenshot displays the CodingGround online Python execution environment. The header includes the CodingGround logo and the text 'Execute Python Online (Python v2.7.13)'. Below the header, there are tabs for 'main.py' and 'STDIN'. The code editor contains the following Python code:

```
1 # Hello World program in Python
2
3 print("Hello World!")
4
5
6
```

The 'Result' pane on the right shows the command '\$python main.py' and the output 'Hello World!'.






- Compile and Execution
- **Overview of C**



# Overview of C: Sample Programs

## • Sample Program 1: Printing a Message

 **codingground**  
SIMPLY EASY CODING

Compile and Execute C Online (GNU GCC v7.1.1)

[Fork](#) [Project](#) [Edit](#) [Setting](#)

[Execute](#) | [Share](#) | [main.c](#) | [STDIN](#)

```
1 #include <stdio.h>
2
3 main()
4 {
5     /* --- printing begins --- */
6     printf("I see, I remember");
7     /* --- printing ends --- */
8 }
9
```

**Result**

```
$gcc -o main *.c
main.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^~~~

$main
I see, I remember
```

Output of this function is **INTEGER**

This function has **two INTEGER** inputs.



```
int KU_add(int a, int b)
{
    return a+b;
}
```

Output of this function is the addition of two inputs.



# Overview of C: Sample Programs

## • Sample Program 1: Printing a Message

 | Compile and Execute C Online (GNU GCC v7.1.1) 

Execute | > Share | main.c | STDIN

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* --- printing begins --- */
6      printf("I see, I remember");
7      /* --- printing ends --- */
8      return 0;
9  }
10
```

**Result**  
\$gcc -o main \*.c  
\$main  
I see, I remember

This will include a header file (name: **stdio.h**).

- **stdio.h** → **header file for printf, ...**  
(if we want to use **printf** function, we have to include this header.)
- Example) math.h (if you want to use complicated math functions.)

IN None

main

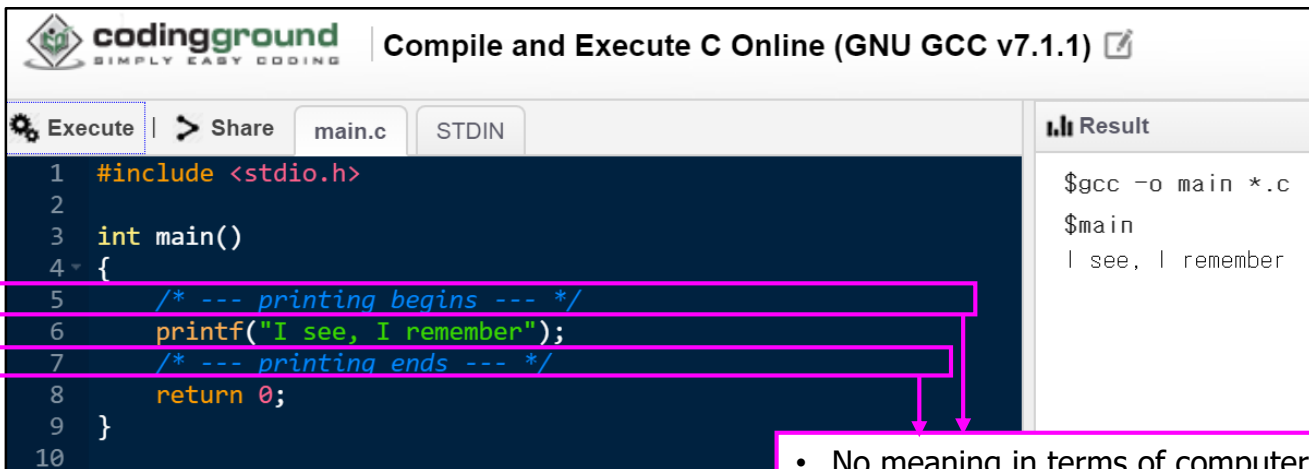
- (Line 5): Print comments (nothing to work)
- **(Line 6): Print the given sentence**
- (Line 7): Print comments (nothing to work)

OUT Int (value: 0)



# Overview of C: Sample Programs

## • Sample Program 1: Printing a Message



```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* --- printing begins --- */
6      printf("I see, I remember");
7      /* --- printing ends --- */
8      return 0;
9  }
10
```

Result

```
$gcc -o main *.c
$main
I see, I remember
```

- No meaning in terms of computers.
- This is required for programmers (for readability).

IN None

main

- (Line 5): Print comments (nothing to work)
- **(Line 6): Print the given sentence**
- (Line 7): Print comments (nothing to work)


OUT


Int (value: 0)



# Overview of C: Sample Programs

- Sample Program 1: Printing a Message
  - Commands (1/2)

 **codingground**  
SIMPLY EASY CODING

Compile and Execute C Online (GNU GCC v7.1.1) 

Execute | Share | main.c | STDIN

```
1  #include <stdio.h>
2
3  int main()
4  {
5      // Korea University, Electrical Engineering
6
7      /*
8         Comment 1
9         Comment 2
10        Comment 3
11       */
12     return 0;
13 }
14
```

**Result**  
\$gcc -o main \*.c  
\$main



# Overview of C: Sample Programs

- Sample Program 1: Printing a Message
  - Commands (2/2)

The screenshot shows the CodingGround online compiler interface. The top bar includes the CodingGround logo, the title "Compile and Execute C Online (GNU GCC v7.1.1)", and navigation links for Fork, Project, Edit, and Setting. Below the bar, there are tabs for "Execute", "Share", "main.c", and "STDIN". The "Execute" tab is active, displaying a C program with the following code:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     /* This /* is */ writing */
6     return 0;
7 }
```

The "Result" tab on the right shows the compilation output, which includes the command and several error messages:

```
$gcc -o main *.c
main.c: In function 'main' :
main.c:5:23: error: unknown type name 'writing'
    /* This /* is */ writing */
                   ^~~~~~
main.c:5:32: error: expected identifier or '(' before '/' token
    /* This /* is */ writing */
                   ^
```



# Overview of C: Sample Programs

## • Sample Program 1: Printing a Message

```
1  #include <stdio.h>
2
3  int main()
4  {
5      /* --- printing begins --- */
6      printf("I see, I remember");
7      /* --- printing ends --- */
8      return 0;
9  }
10
```

Result

```
$gcc -o main *.c
$main
I see, I remember
```

**printf()** function

- Print out the given string.

IN None

main

- (Line 5): Print comments (nothing to work)
- **(Line 6): Print the given sentence**
- (Line 7): Print comments (nothing to work)

OUT

Int (value: 0)



- Sample Program 1: Printing a Message
  - **printf** function

<div>⚙ Execute   ➤ Share</div> <div>main.c    STDIN</div> <pre>1  #include &lt;stdio.h&gt; 2 3  int main() 4  { 5      printf("My name is "); 6      printf("Joongheon Kim.\n"); 7      printf("\n"); 8      printf("I am a professor\n at Korea Univ.\n"); 9      return 0; 10 }</pre>	<div>📊 Result</div> <pre>\$gcc -o main *.c \$main My name is Joongheon Kim.  I am a professor at Korea Univ.</pre>
---	--





- Sample Program 2: Adding Two Numbers

Execute | Share

main.c

STDIN

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int number;
6      float amount;
7
8      number = 100;
9      amount = 30.75 + 75.35;
10
11     printf("%d\n", number);
12     printf("%5.2f\n", amount);
13 }
14
```

Result

```
$gcc -o main *.c
$main
100
106.10
```



## • Sample Program 2: Adding Two Numbers

Execute | Share

main.c

STDIN

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int number;
6      float amount;
7
8      number = 100;
9      amount = 30.75 + 75.35;
10
11     printf("%d\n", number);
12     printf("%5.2f\n", amount);
13 }
14
```

Declaration: two numbers



- Variable (integer type): number
- Variable (float type): amount

Result


```
$gcc -o main *.c
$main
100
106.10
```



## • Sample Program 2: Adding Two Numbers

 Execute |  Share | main.c | STDIN

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int number;
6      float amount;
7
8      number = 100;
9      amount = 30.75 + 75.35;
10
11     printf("%d\n", number);
12     printf("%5.2f\n", amount);
13 }
14
```

 Result

```
$gcc -o main *.c
$main
100
106.10
```

Assignment

- Assign **100** into **number**.
- Assign the result of **30.75+75.35** into **amount**.



## • Sample Program 2: Adding Two Numbers

Execute   > Share	main.c	STDIN	Result
<pre>1  #include &lt;stdio.h&gt; 2 3  int main() 4  { 5      int number; 6      float amount; 7 8      number = 100; 9      amount = 30.75 + 75.35; 10 11     printf("%d\n", number); 12     printf("%.2f\n", amount); 13 } 14</pre>			<p>\$gcc -o main *.c</p> <p>\$main</p> <p>100</p> <p>106.10</p>

Print the **number** and **amount** values.

- Print the value of number (**integer** with **%d**).
- Print the value of amount (**float** with **%5.2f**).
  - 5 places in all, 2 places to the right of the decimal point.

```
printf("%d\n", number);
printf("%.2f\n", amount);
```



## • Sample Program 3: Interest Calculation

Execute   > Share	main.c	STDIN	Result
<pre>1  #include &lt;stdio.h&gt; 2 3  #define PERIOD 10 4  #define PRINCIPAL 5000.00 5 6  int main(){ 7      int year; 8      float amount, value, interest_rate; 9 10     amount = PRINCIPAL; 11     interest_rate = 0.11; 12     year = 0; 13 14     while(year &lt;= PERIOD){ 15         printf("%2d %8.2f\n", year, amount); 16         value = amount + interest_rate * amount; 17         year = year + 1; 18         amount = value; 19     } 20     return 0; 21 }</pre>			<pre>\$gcc -o main *.c \$main 0 5000.00 1 5550.00 2 6160.50 3 6838.15 4 7590.35 5 8425.29 6 9352.07 7 10380.80 8 11522.69 9 12790.18 10 14197.10</pre>



## • Sample Program 3: Interest Calculation

```
Execute | > Share | main.c | STDIN | Result
1  #include <stdio.h>
2
3  #define PERIOD 10
4  #define PRINCIPAL 5000.00
5
6  int main(){
7      int year;
8      float amount, value, interest_rate;
9
10     amount = PRINCIPAL;
11     interest_rate = 0.11;
12     year = 0;
13
14     while(year <= PERIOD){
15         printf("%2d %8.2f\n", year, amount);
16         value = amount + interest_rate * amount;
17         year = year + 1;
18         amount = value;
19     }
20     return 0;
21 }
```

\$gcc -o main \*.c

2	5100.00
3	6838.15
4	7590.35
5	8425.29
6	9352.07
7	10380.80
8	11522.69
9	12790.18
10	14197.10

In the given entire code,

- **PERIOD** means 10
- **PRINCIPAL** means 5000.00



## • Sample Program 3: Interest Calculation

Execute | Share

main.c | STDIN

```
1 #include <stdio.h>
2
3 #define PERIOD 10
4 #define PRINCIPAL 5000.00
5
6 int main(){
7     int year;
8     float amount, value, interest_rate;
9
10    amount = PRINCIPAL;
11    interest_rate = 0.11;
12    year = 0;
13
14    while(year <= PERIOD){
15        printf("%2d %8.2f\n", year, amount);
16        value = amount + interest_rate * amount;
17        year = year + 1;
18        amount = value;
19    }
20    return 0;
21 }
```

Result

```
$gcc -o main *.c
$main
0 5000.00
1 5550.00
2 6160.50
7 10380.80
8 11522.69
9 12790.18
10 14197.10
```

### Variable Declaration

- Integer: year
- Float: amount, value, interest\_rate



## • Sample Program 3: Interest Calculation

Execute | Share

main.c | STDIN

Result

```
1  #include <stdio.h>
2
3  #define PERIOD 10
4  #define PRINCIPAL 5000.00
5
6  int main(){
7      int year;
8      float amount, value, interest_rate;
9
10     amount = PRINCIPAL;
11     interest_rate = 0.11;
12     year = 0;
13
14     while(year <= PERIOD){
15         printf("%2d %8.2f\n", year, amount);
16         value = amount + interest_rate * amount;
17         year = year + 1;
18         amount = value;
19     }
20     return 0;
21 }
```

\$gcc -o main \*.c

\$main

0	5000.00
1	5550.00
2	6160.50
3	6838.15
4	7590.35
5	8425.29

### Calculation

- Assign the value of PRINCIPAL (i.e., 5000.00) into **amount (float)**
- Assign 0.11 into **interest\_rate (float)**
- Assign 0 into **year (integer)**





## • Sample Program 3: Interest Calculation

Execute | Share | main.c | STDIN

```
1 #include <stdio.h>
2
3 #define PERIOD 10
4 #define PRINCIPAL 5000.00
5
6 int main(){
7     int year;
8     float amount, value, interest_rate;
9
10    amount = PRINCIPAL;
11    interest_rate = 0.11;
12    year = 0;
13
14    while(year <= PERIOD){
15        printf("%2d %8.2f\n", year, amount);
16        value = amount + interest_rate * amount;
17        year = year + 1;
18        amount = value;
19    }
20    return 0;
21 }
```

Result

```
$gcc -o main *.c
$main
0 5000.00
1 5550.00
2 6160.50
3 6838.15
4 7590.35
5 8425.29
6 9352.07
7 10380.80
8 11522.69
9 12790.18
10 14197.10
```

- Loop: **while(condition){A}**
- If the given **condition** is true, **A** will be executed.
  - (line 15) print **year** and **amount** values.
  - (line 16) compute **value**.
  - (line 17) increase **year**.
  - (line 18) update **amount** (i.e., interest added)



- Sample Program 4: Use of Subroutines

Execute   > Share	main.c	STDIN	Result
<pre>1  #include &lt;stdio.h&gt; 2 3  int mul(int a, int b); 4 5  int main(){ 6      int a, b, c; 7      a = 5; 8      b = 10; 9      c = mul(a, b); 10     printf("multiplication of %d and %d is %d.", a, b, c); 11     return 0; 12 } 13 14 int mul(int x, int y){ 15     int p; 16     p = x*y; 17     return p; 18 }</pre>			<pre>\$gcc -o main *.c \$main multiplication of 5 and 10 is 50.</pre>



## • Sample Program 4: Use of Subroutines

Execute | > Share | main.c | STDIN

```
1  #include <stdio.h>
2
3  int mul(int a, int b);
4
5  int main(){
6      int a, b, c;
7      a = 5;
8      b = 10;
9      c = mul(a, b);
10     printf("multiplication of %d and %d is %d.", a, b, c);
11     return 0;
12 }
13
14 int mul(int x, int y){
15     int p;
16     p = x*y;
17     return p;
18 }
```

Result

Function Declaration (name: **mul**)

- Input: two INTEGER values, **a** and **b**
- Output: one INTEGER value

50 is 50.



## • Sample Program 4: Use of Subroutines

Execute   > Share	main.c	STDIN	Result
<pre>1  #include &lt;stdio.h&gt; 2 3  int mul(int a, int b); 4 5  int main(){ 6      int a, b, c; 7      a = 5; 8      b = 10; 9      c = mul(a, b); 10     printf("multiplication of %d and %d is %d.", a, b, c); 11     return 0; 12 } 13 14 int mul(int x, int y){ 15     int p; 16     p = x*y; 17     return p; 18 }</pre>			<pre>\$gcc -o main *.c \$main multiplication of 5 and 10 is 50.</pre>

Variable **c** Calculation

- The value of **c** will be the output of function **mul()** when the two input values are **a** and **b**.



## • Sample Program 4: Use of Subroutines

Execute   > Share	main.c	STDIN	Result
<pre>1  #include &lt;stdio.h&gt; 2 3  int mul(int a, int b); 4 5  int main(){ 6      int a, b, c; 7      a = 5; 8      b = 10; 9      c = mul(a, b); 10     printf("multiplication of %d and %d is %d.", a, b, c); 11     return 0; 12 } 13 14 int mul(int x, int y){ 15     int p; 16     p = x*y; 17     return p; 18 }</pre>			<pre>\$gcc -o main *.c \$main multiplication of 5 and 10 is 50.</pre>

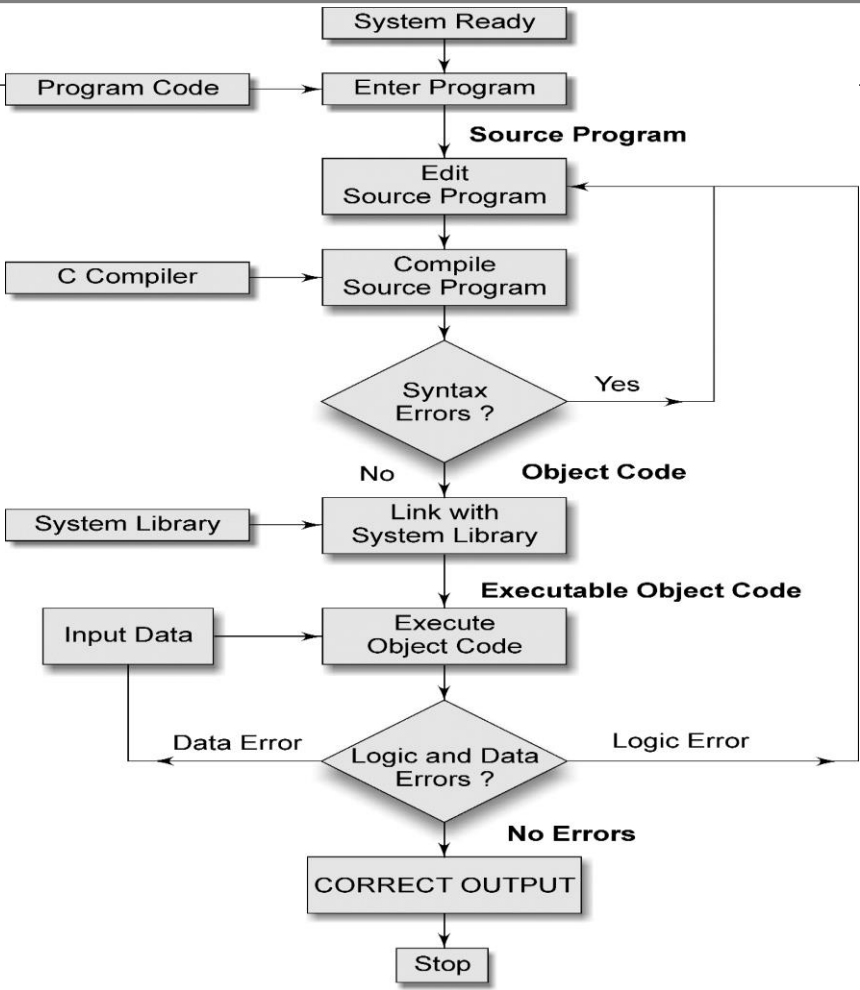
Function **mul()**

- Inputs: **x** and **y** (don't need to be **a** and **b**)
- (line 15) integer variable declaration **p**
- (line 16) **p** has the value of **a\*b**
- (line 17) the **p** is the output of this function.



# Overview of C

- C Program Running Sequence





- Sample Program 1: Printing a Message
- Sample Program 2: Adding Two Numbers
- Sample Program 3: Interest Calculation
- Sample Program 4: Use of Subroutines



- Sample Program 1: Printing a Message

Execute | Share | main.py | STDIN | Login x

```
1 # This is Python
2 '''
3 This is Python.
4 Pythin is simple and powerful.
5 '''
6 print("I see, I remember")
7
```

Result

\$python main.py  
I see, I remember



codingground | Compile and Execute C Online (GNU GCC v7.1.1)

Fork | Project | Edit | Setting

Execute | Share | main.c | STDIN

```
1 #include <stdio.h>
2
3 main()
4 {
5     /* --- printing begins --- */
6     printf("I see, I remember");
7     /* --- printing ends --- */
8 }
9
```

Result

\$gcc -o main \*.c  
main.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]  
main()  
^~~~  
\$main  
I see, I remember








- Sample Program 2: Adding Two Numbers

 Execute    Share	main.py	STDIN	Login x	 Result
<pre>1 number = 100 2 amount = 30.75 + 75.35 3 print(number) 4 print(amount)</pre>				<pre>\$python main.py 100 106.1</pre>



 Execute    Share	main.c	STDIN		 Result
<pre>1 #include &lt;stdio.h&gt; 2 3 int main() 4 { 5     int number; 6     float amount; 7 8     number = 100; 9     amount = 30.75 + 75.35; 10 11     printf("%d\n", number); 12     printf("%5.2f\n", amount); 13 } 14</pre>				<pre>\$gcc -o main *.c \$main 100 106.10</pre>



## • Sample Program 3: Interest Calculation

Execute   > Share	main.py	STDIN	Login x	Result
<pre>1 PERIOD = 10 2 PRINCIPAL = 5000.00 3 amount = PRINCIPAL 4 interest_rate = 0.11 5 year = 0 6 while (year &lt;= PERIOD): 7     print(year, amount) 8     value = amount + interest_rate * amount 9     year = year + 1 10    amount = value 11 12</pre>				<pre>\$python main.py (0, 5000.0) (1, 5550.0) (2, 6160.5) (3, 6838.155) (4, 7590.352049999995) (5, 8425.2907755) (6, 9352.072760805) (7, 10380.80076449355) (8, 11522.68884858784) (9, 12790.184621932503) (10, 14197.104930345078)</pre>



Execute   > Share	main.c	STDIN	Result	
<pre>1 #include &lt;stdio.h&gt; 2 3 #define PERIOD 10 4 #define PRINCIPAL 5000.00 5 6 int main(){ 7     int year; 8     float amount, value, interest_rate; 9 10    amount = PRINCIPAL; 11    interest_rate = 0.11; 12    year = 0; 13 14    while(year &lt;= PERIOD){ 15        printf("%2d %8.2f\n", year, amount); 16        value = amount + interest_rate * amount; 17        year = year + 1; 18        amount = value; 19    } 20    return 0; 21 }</pre>				<pre>\$gcc -o main *.c \$main 0 5000.00 1 5550.00 2 6160.50 3 6838.15 4 7590.35 5 8425.29 6 9352.07 7 10380.80 8 11522.69 9 12790.18 10 14197.10</pre>



## • Sample Program 4: Use of Subroutines

<div>Execute   Share   main.py   STDIN   Login x</div> <pre>1 def mul(x, y): 2     return x*y 3 a = 5 4 b = 10 5 c = mul(a, b) 6 print("multiplicaiton of ", a, " and ", b, " is ", c, ".")</pre>	<div>Result</div> <pre>\$python main.py ('multiplicaiton of ', 5, ' and ', 10, ' is ', 50, '.')</pre>
---	---



<div>Execute   Share   main.c   STDIN</div> <pre>1 #include &lt;stdio.h&gt; 2 3 int mul(int a, int b); 4 5 int main(){ 6     int a, b, c; 7     a = 5; 8     b = 10; 9     c = mul(a, b); 10    printf("multiplication of %d and %d is %d.", a, b, c); 11    return 0; 12 } 13 14 int mul(int x, int y){ 15     int p; 16     p = x*y; 17     return p; 18 }</pre>	<div>Result</div> <pre>\$gcc -o main *.c \$main multiplication of 5 and 10 is 50.</pre>
--	---



## • Sample Program 4: Use of Subroutines

The screenshot shows the Spyder Python IDE interface. The editor window displays a Python script named `untitled3.py` with the following code:

```
1 def mul(x, y):
2     return x*y
3 a = 5
4 b = 10
5 c = mul(a, b)
6 print("multiplicaiton of ", a, " and ", b, " is ", c, ".")
7
8
```

The IPython console on the right shows the output of running the script:

```
Python 3.6.5 [Anaconda, Inc.] (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

Restarting kernel...

In [1]: runfile('C:/Users/CAU/Desktop/untitled3.py', wdir='C:/Users/CAU/Desktop')
multiplicaiton of 5 and 10 is 50 .
```