2021-11-07

1)OPERATORS

2)FUNCTION

3)EXERCISES

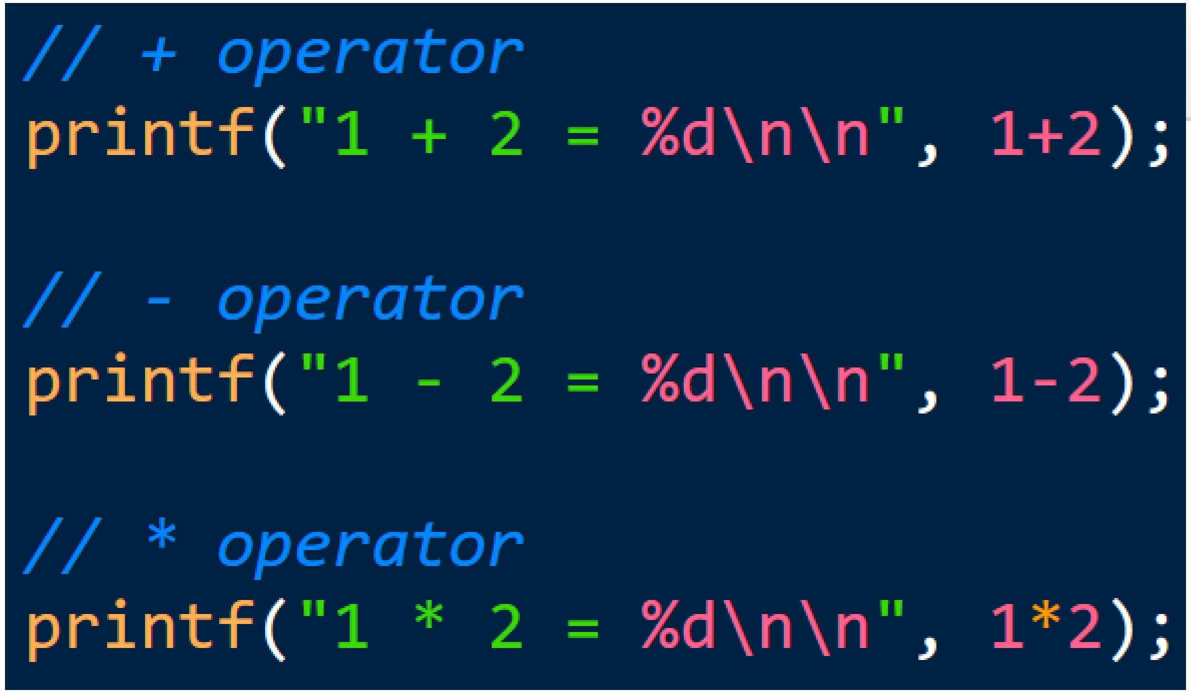
1. OPERATORS

1.1. ARITHMETIC OPERATORS

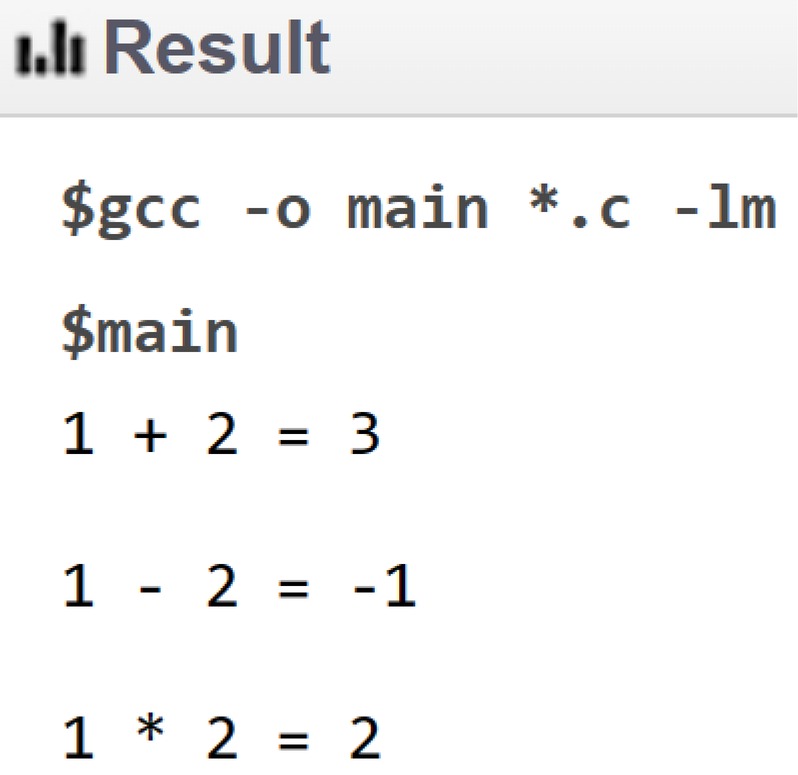
Operator	Description
+	Add two operands
-	Substract second operand from the first
*	Multiplies two operands
/	Divides numerator by de-numerator
% (Modulus Operator)	Get remainder of an integer division
++	Increases the integer value by one
--	Decreases the integer value by one

1. OPERATORS

1.1. ARITHMETIC OPERATORS



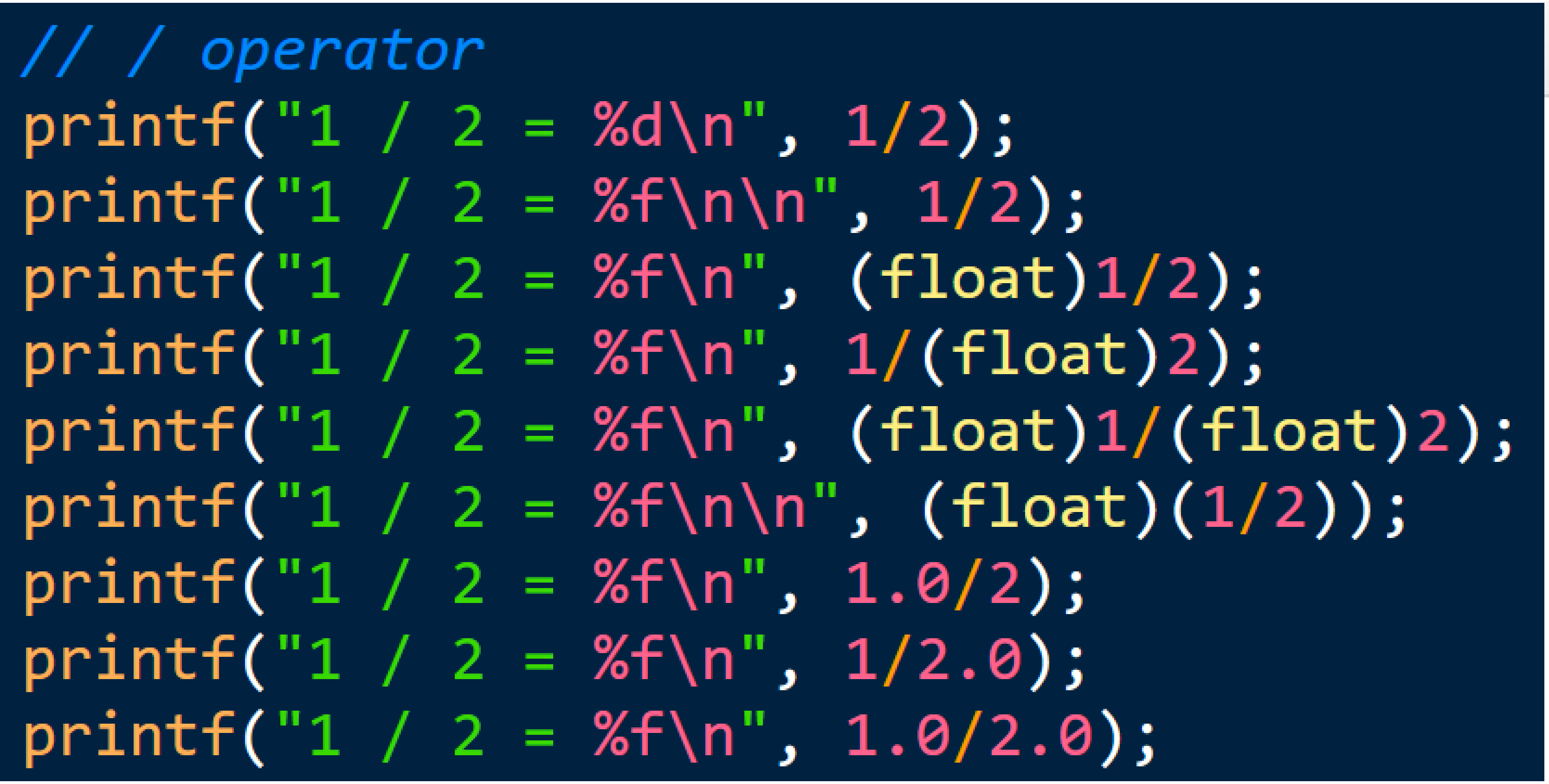
```
// + operator
printf("1 + 2 = %d\n\n", 1+2);

// - operator
printf("1 - 2 = %d\n\n", 1-2);

// * operator
printf("1 * 2 = %d\n\n", 1*2);
```



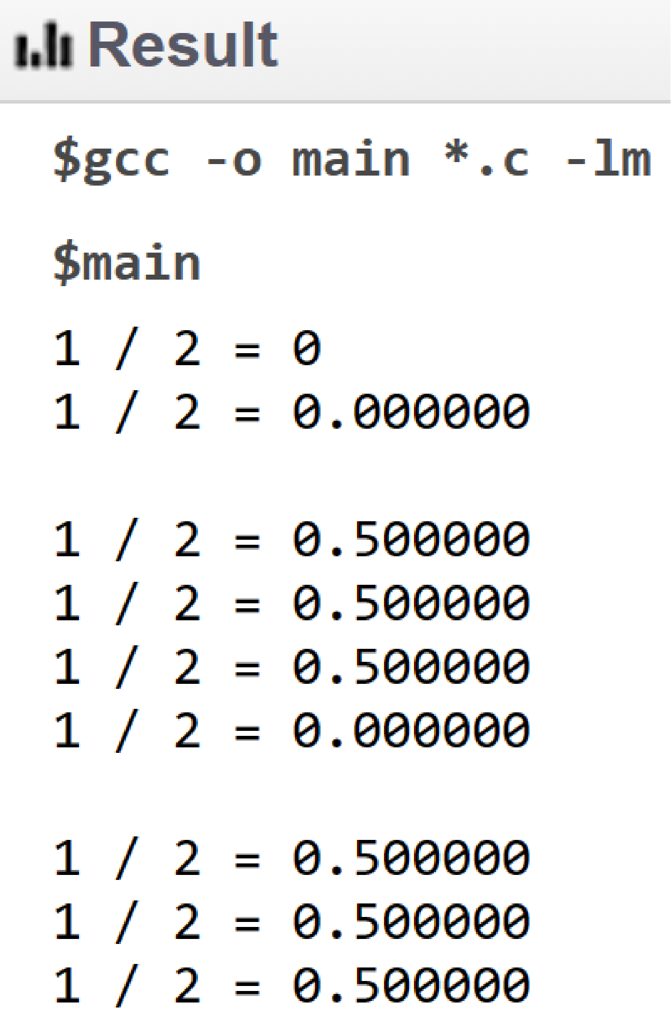
```
$gcc -o main *.c -lm
$main
1 + 2 = 3

1 - 2 = -1

1 * 2 = 2
```

1. OPERATORS

1.1. ARITHMETIC OPERATORS



```c
// / operator
printf("1 / 2 = %d\n", 1/2);
printf("1 / 2 = %f\n\n", 1/2);
printf("1 / 2 = %f\n", (float)1/2);
printf("1 / 2 = %f\n", 1/(float)2);
printf("1 / 2 = %f\n", (float)1/(float)2);
printf("1 / 2 = %f\n\n", (float)(1/2));
printf("1 / 2 = %f\n", 1.0/2);
printf("1 / 2 = %f\n", 1/2.0);
printf("1 / 2 = %f\n", 1.0/2.0);
```



```
$gcc -o main *.c -lm

$main
1 / 2 = 0
1 / 2 = 0.000000

1 / 2 = 0.500000
1 / 2 = 0.500000
1 / 2 = 0.500000
1 / 2 = 0.000000

1 / 2 = 0.500000
1 / 2 = 0.500000
1 / 2 = 0.500000
```

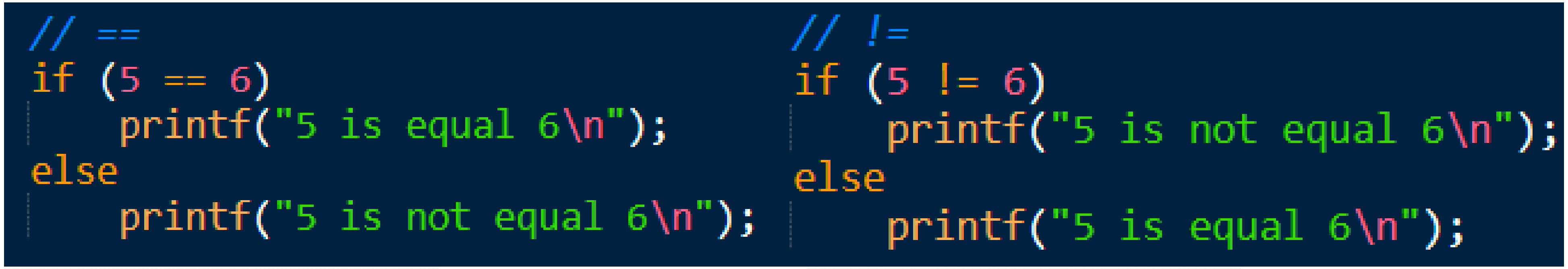
1. OPERATORS

1.2. RELATIONAL OPERATORS

Operator	Description
==	Return true if 2 operands are equal
!=	Return true if 2 operands are not equal
>	Return true if the left operand is greater than the right one
<	Return true if the left operand is less than the right one
>=	Return true if the left operand is greater than or equal to the right one
<=	Return true if the left operand is less than or equal to the right one

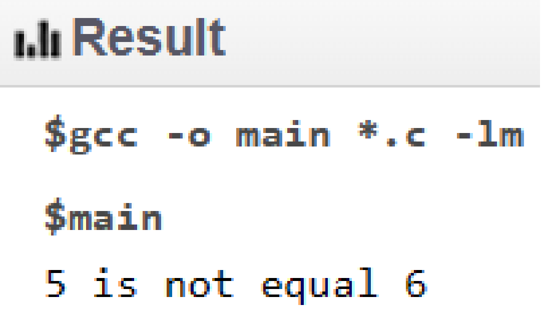
1. OPERATORS

1.2. RELATIONAL OPERATORS



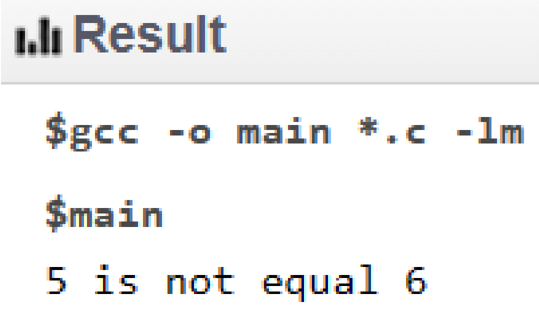
```c
// ==
if (5 == 6)
    printf("5 is equal 6\n");
else
    printf("5 is not equal 6\n");
```

```c
// !=
if (5 != 6)
    printf("5 is not equal 6\n");
else
    printf("5 is equal 6\n");
```



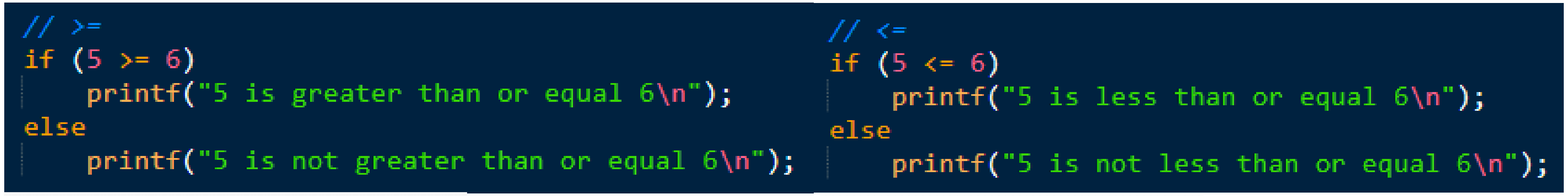
Result

```
$gcc -o main *.c -lm
$main
5 is not equal 6
```



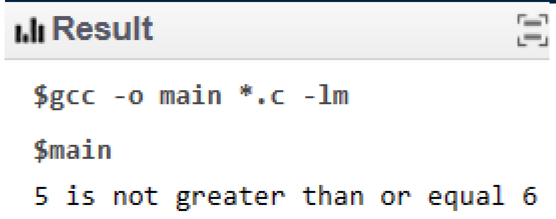
Result

```
$gcc -o main *.c -lm
$main
5 is not equal 6
```



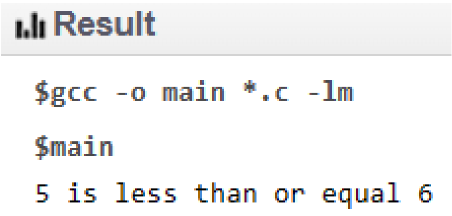
```c
// >=
if (5 >= 6)
    printf("5 is greater than or equal 6\n");
else
    printf("5 is not greater than or equal 6\n");
```

```c
// <=
if (5 <= 6)
    printf("5 is less than or equal 6\n");
else
    printf("5 is not less than or equal 6\n");
```



Result

```
$gcc -o main *.c -lm
$main
5 is not greater than or equal 6
```



Result

```
$gcc -o main *.c -lm
$main
5 is less than or equal 6
```

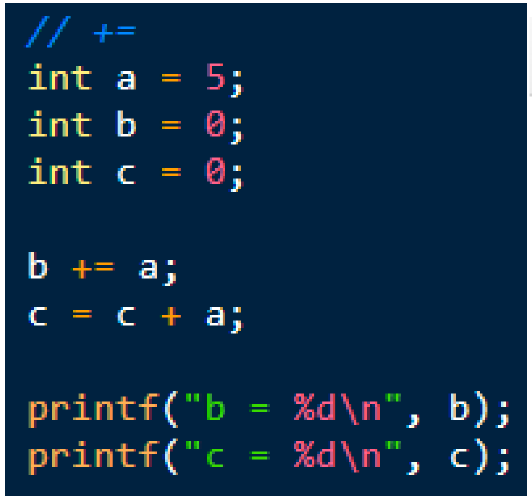
1. OPERATORS

1.3. ASSIGNMENT OPERATORS

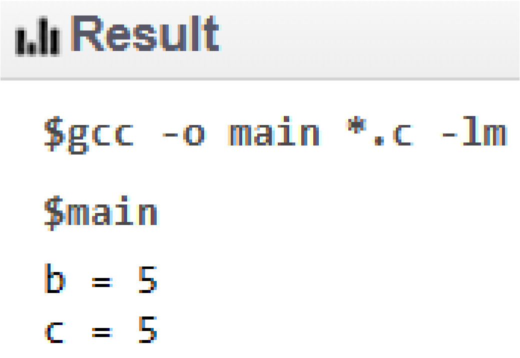
Operator	Description
+=	C += A ⬄ C = C + A
-=	C -= A ⬄ C = C – A
*=	C *= A ⬄ C = C * A
/=	C /= A ⬄ C = C / A
%=	C %= A ⬄ C = C % A

1. OPERATORS

1.3. ASSIGNMENT OPERATORS

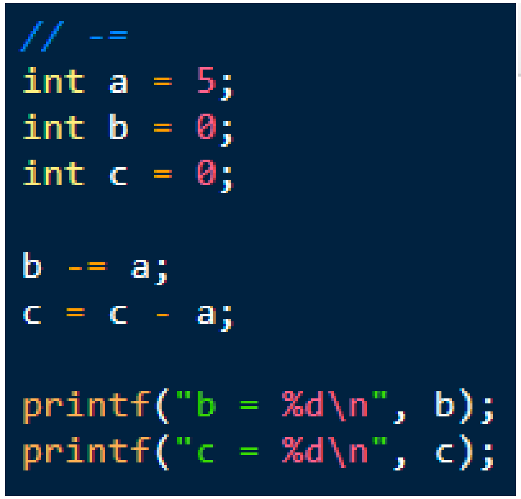


```
// +=
int a = 5;
int b = 0;
int c = 0;

b += a;
c = c + a;

printf("b = %d\n", b);
printf("c = %d\n", c);
```

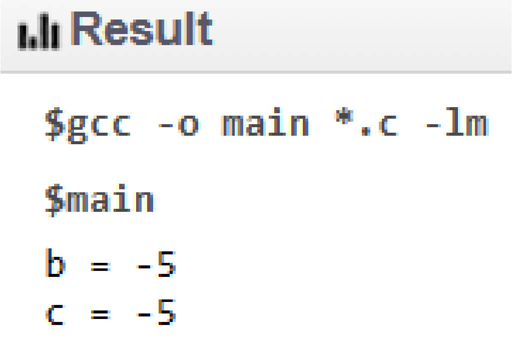


Result

```
$gcc -o main *.c -lm

$main

b = 5
c = 5
```

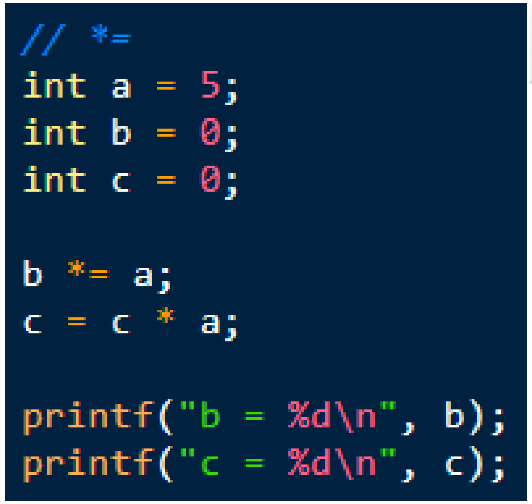


```
// -=
int a = 5;
int b = 0;
int c = 0;

b -= a;
c = c - a;

printf("b = %d\n", b);
printf("c = %d\n", c);
```

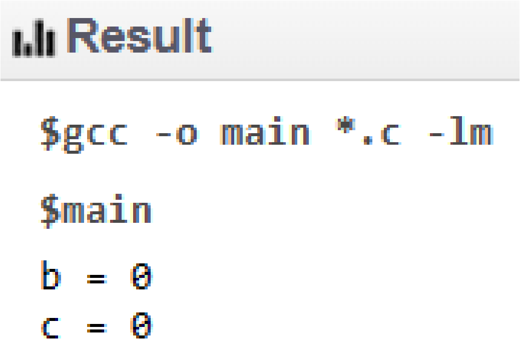


Result

```
$gcc -o main *.c -lm

$main

b = -5
c = -5
```

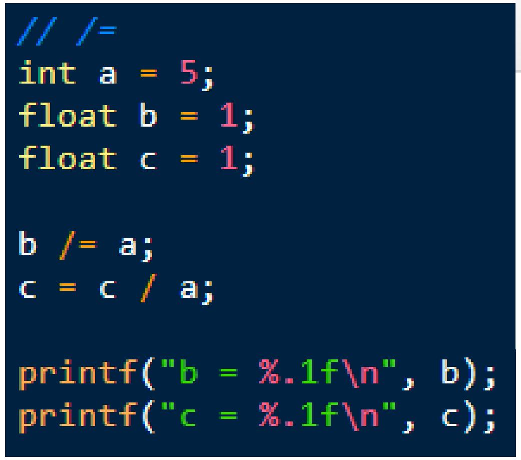


```
// *=
int a = 5;
int b = 0;
int c = 0;

b *= a;
c = c * a;

printf("b = %d\n", b);
printf("c = %d\n", c);
```

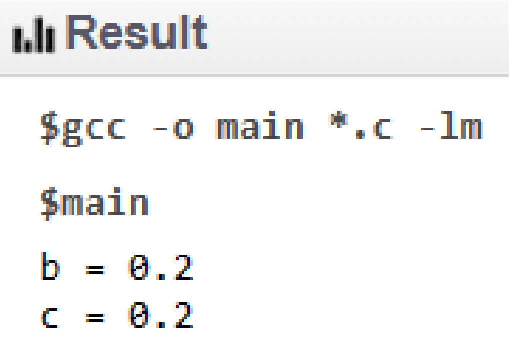


Result

```
$gcc -o main *.c -lm

$main

b = 0
c = 0
```



```
// /=
int a = 5;
float b = 1;
float c = 1;

b /= a;
c = c / a;

printf("b = %.1f\n", b);
printf("c = %.1f\n", c);
```

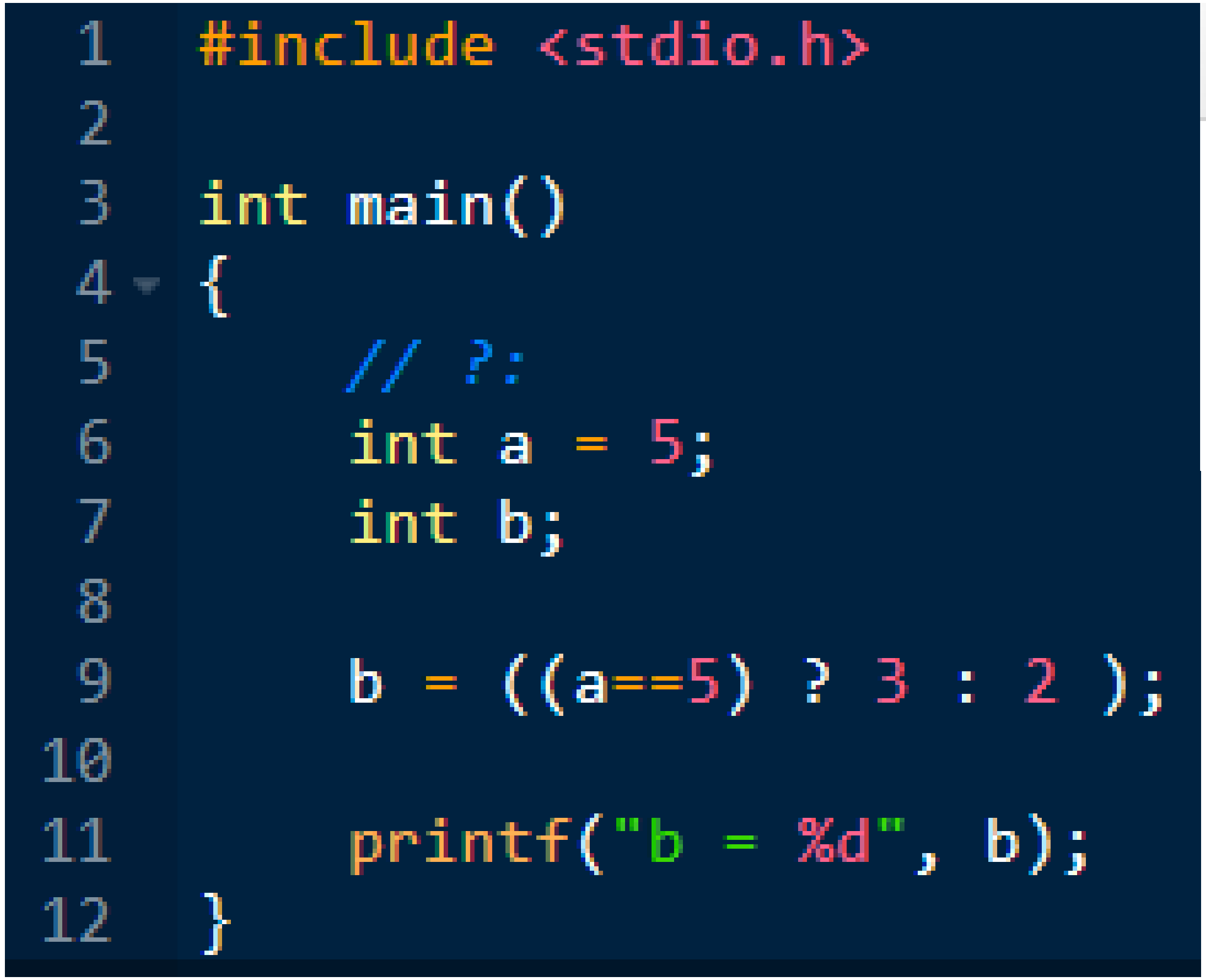


Result

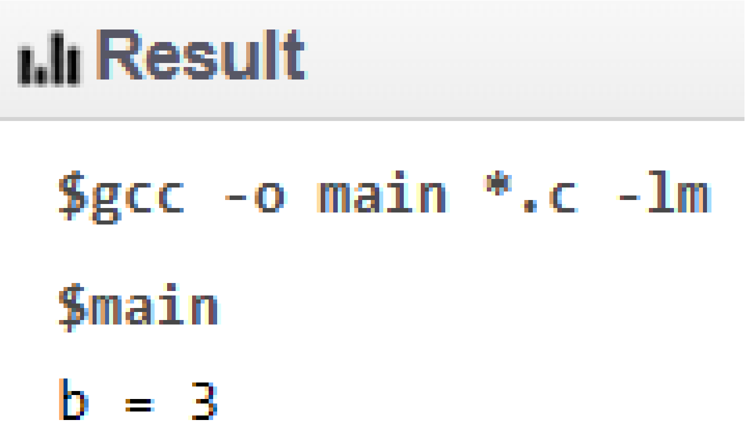
```
$gcc -o main *.c -lm

$main

b = 0.2
c = 0.2
```

1. OPERATORS

1.4. CONDITIONAL EXPRESSION OPERATORS



```c
#include <stdio.h>

int main()
{
      // ?:
      int a = 5;
      int b;

      b = ((a==5) ? 3 : 2 );

      printf("b = %d", b);
}
```

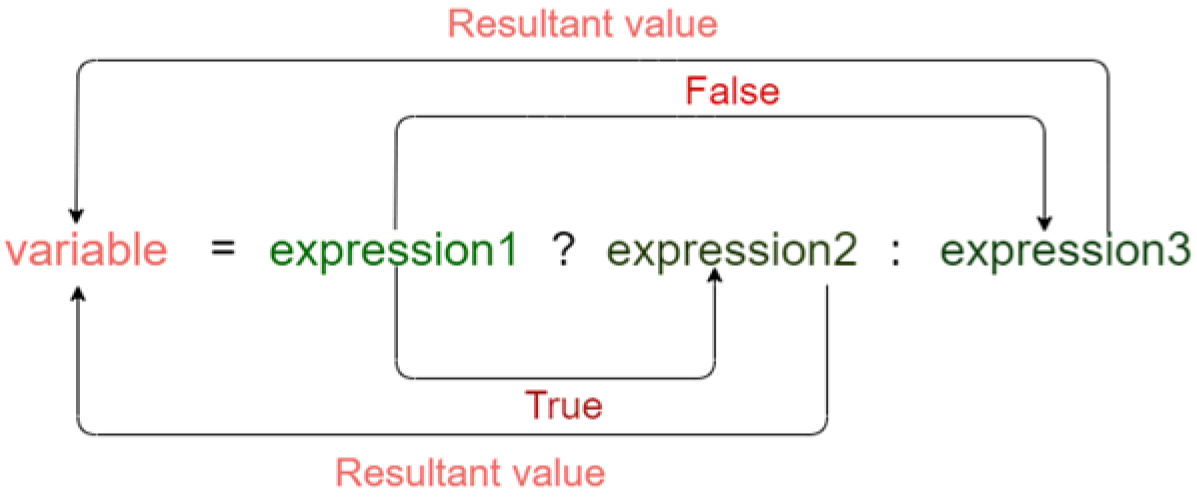


```
$gcc -o main *.c -lm

$main

b = 3
```

1. OPERATORS

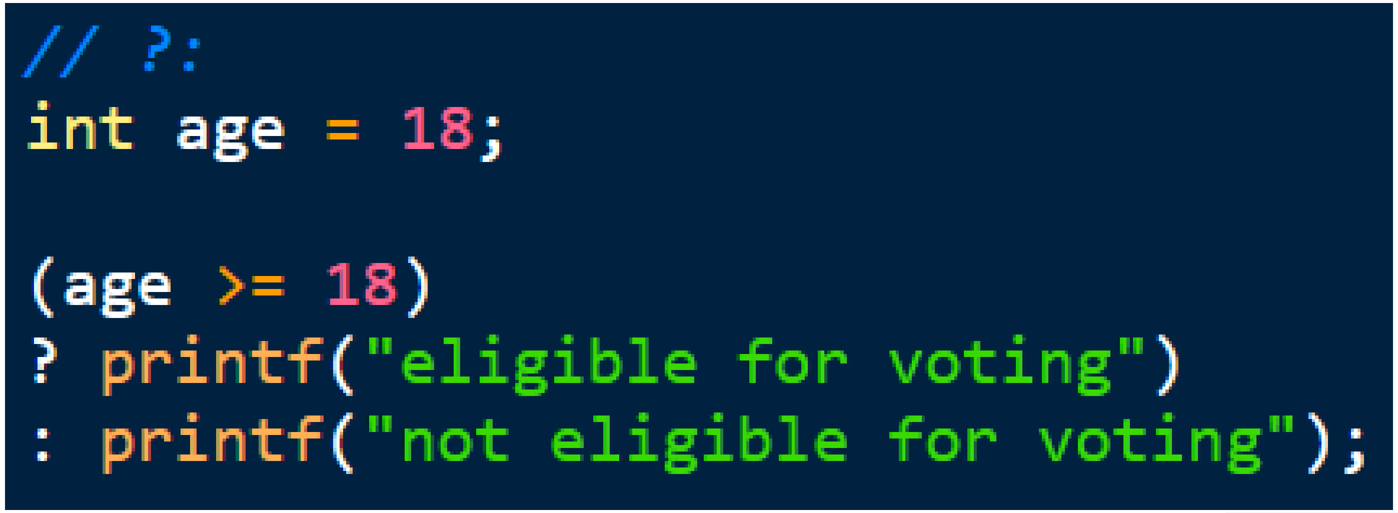
1.4. CONDITIONAL EXPRESSION OPERATORS

Operator	Description
?:	Is condition is true ? Then X : otherwise Y

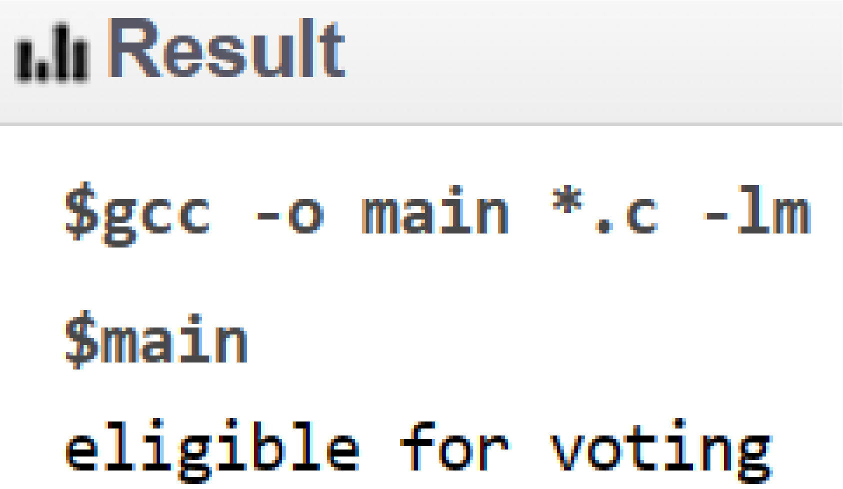


1. OPERATORS

1.4. CONDITIONAL EXPRESSION OPERATORS

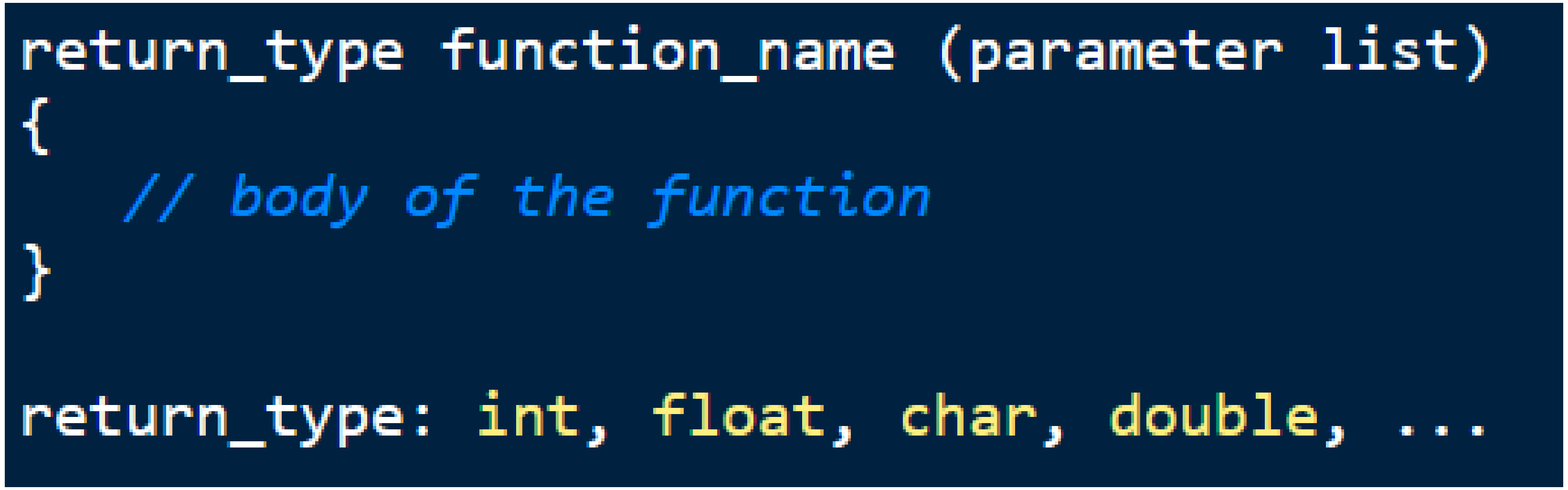


```c
// ?:
int age = 18;

(age >= 18)
? printf("eligible for voting")
: printf("not eligible for voting");
```



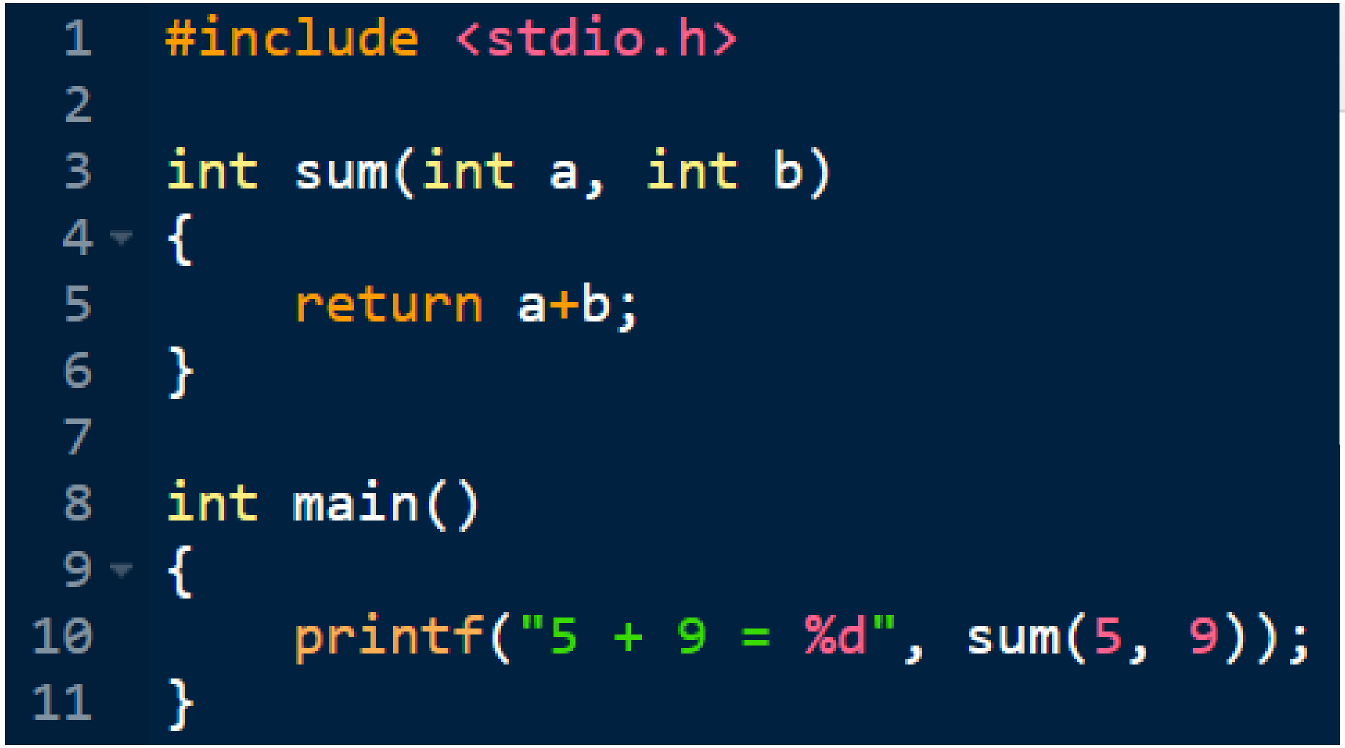
```
$gcc -o main *.c -lm

$main
eligible for voting
```

2. FUNCTION

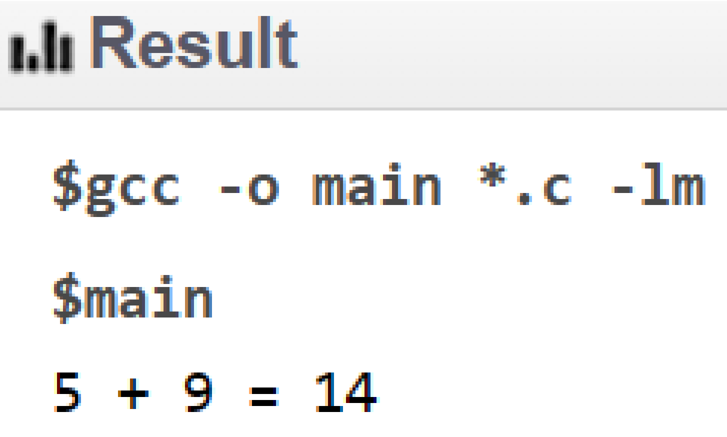


```
return_type function_name (parameter list)
{
   // body of the function
}

return_type: int, float, char, double, ...
```

2. FUNCTION



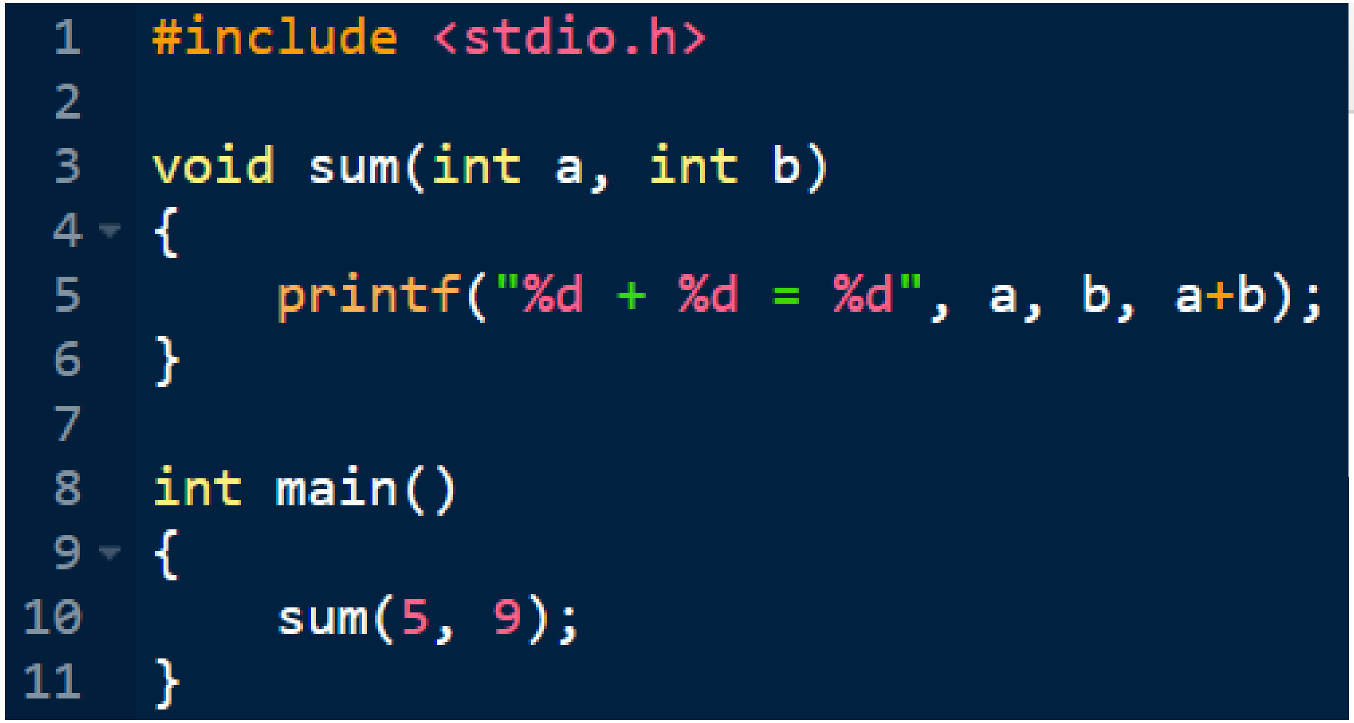
```c
#include <stdio.h>

int sum(int a, int b)
{
	return a+b;
}

int main()
{
	printf("5 + 9 = %d", sum(5, 9));
}
```



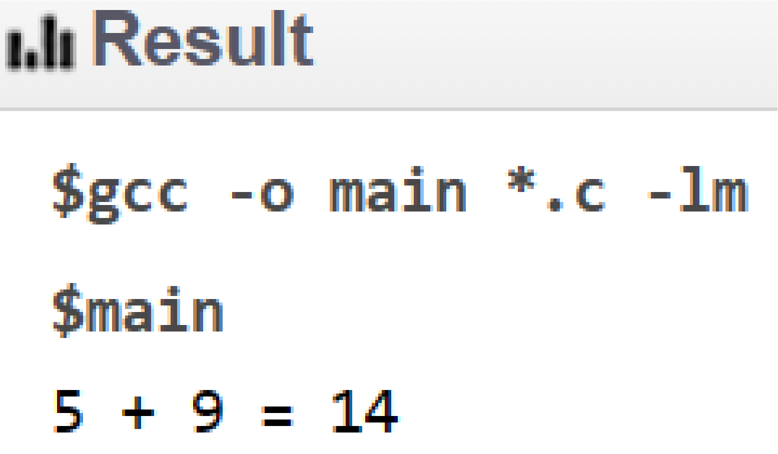
Result

```
$gcc -o main *.c -lm

$main

5 + 9 = 14
```

2. FUNCTION



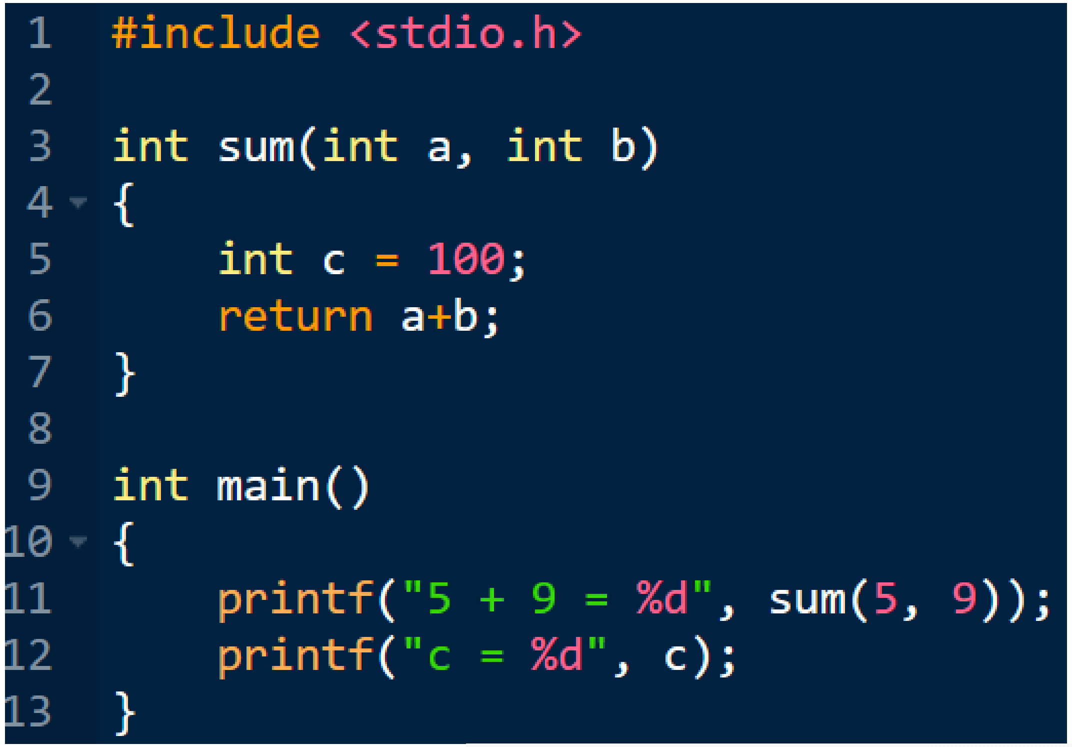
```c
#include <stdio.h>

void sum(int a, int b)
{
    printf("%d + %d = %d", a, b, a+b);
}

int main()
{
    sum(5, 9);
}
```



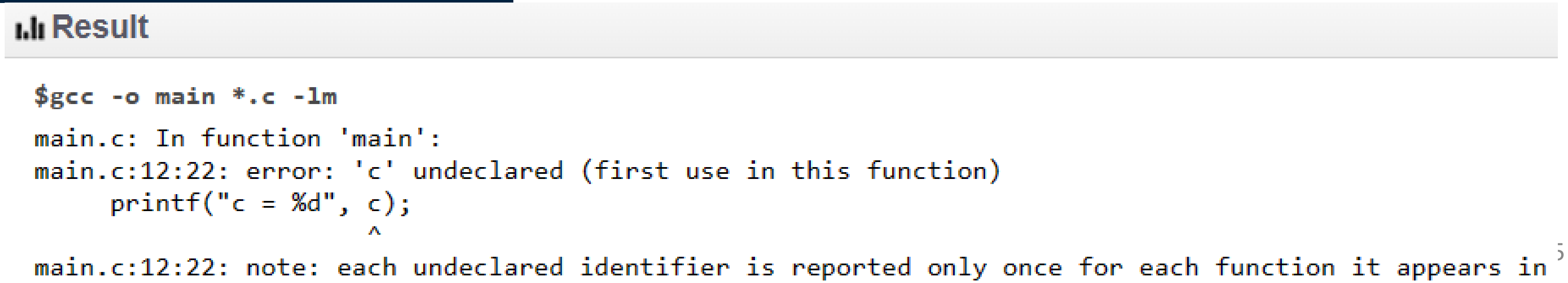
Result

```
$gcc -o main *.c -lm

$main

5 + 9 = 14
```

2. FUNCTION

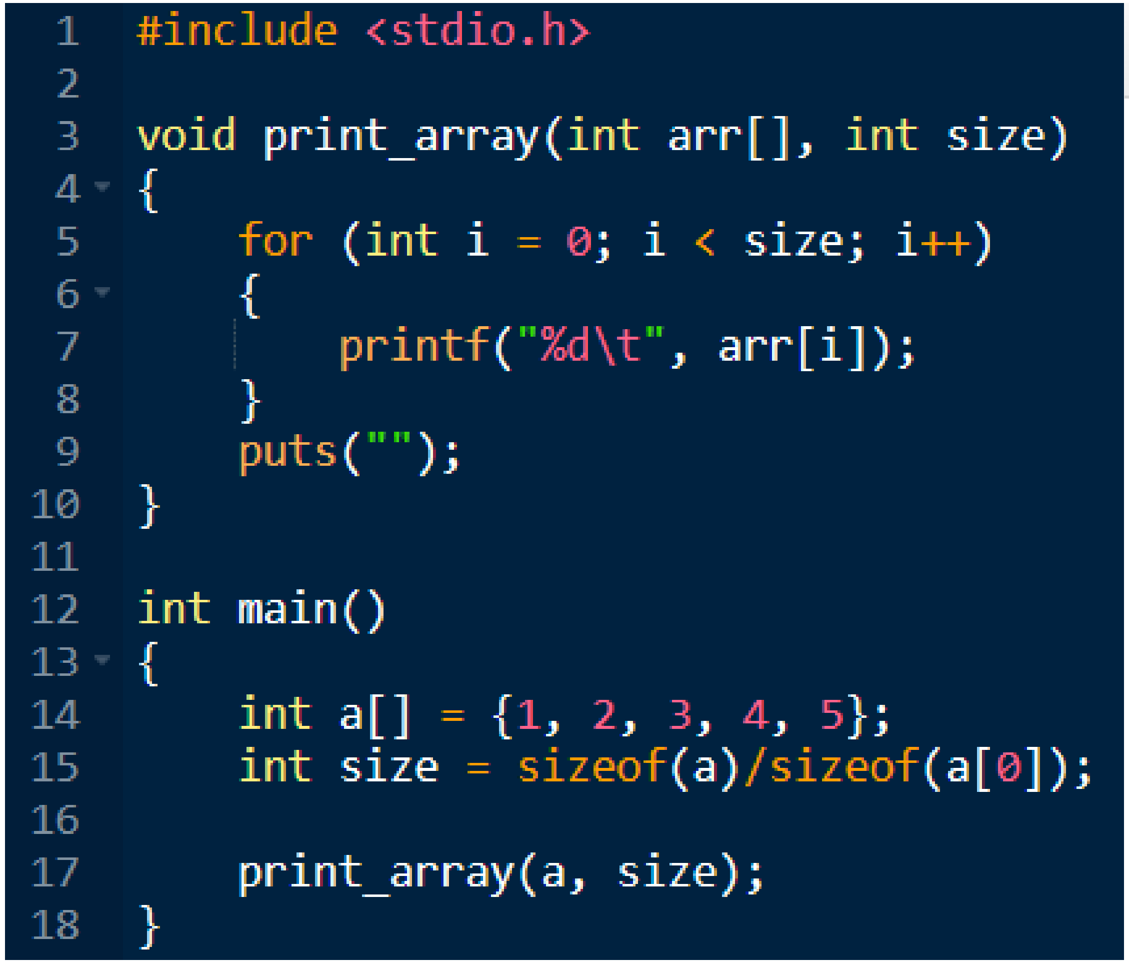


```c
#include <stdio.h>

int sum(int a, int b)
{
    int c = 100;
    return a+b;
}

int main()
{
    printf("5 + 9 = %d", sum(5, 9));
    printf("c = %d", c);
}
```

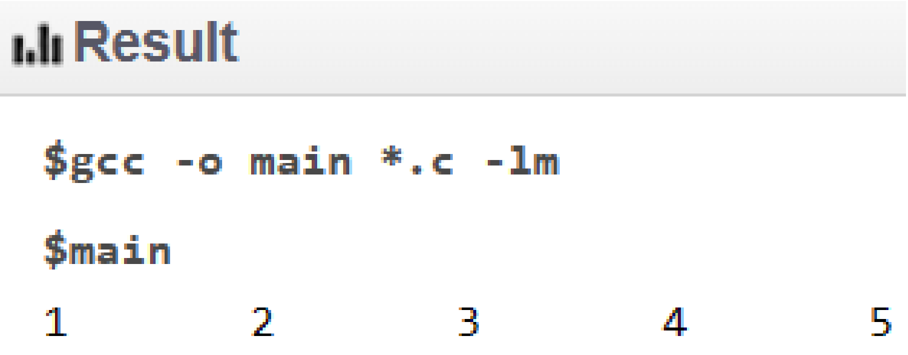


```
$gcc -o main *.c -lm
main.c: In function 'main':
main.c:12:22: error: 'c' undeclared (first use in this function)
     printf("c = %d", c);
                      ^
main.c:12:22: note: each undeclared identifier is reported only once for each function it appears in
```

2. FUNCTION



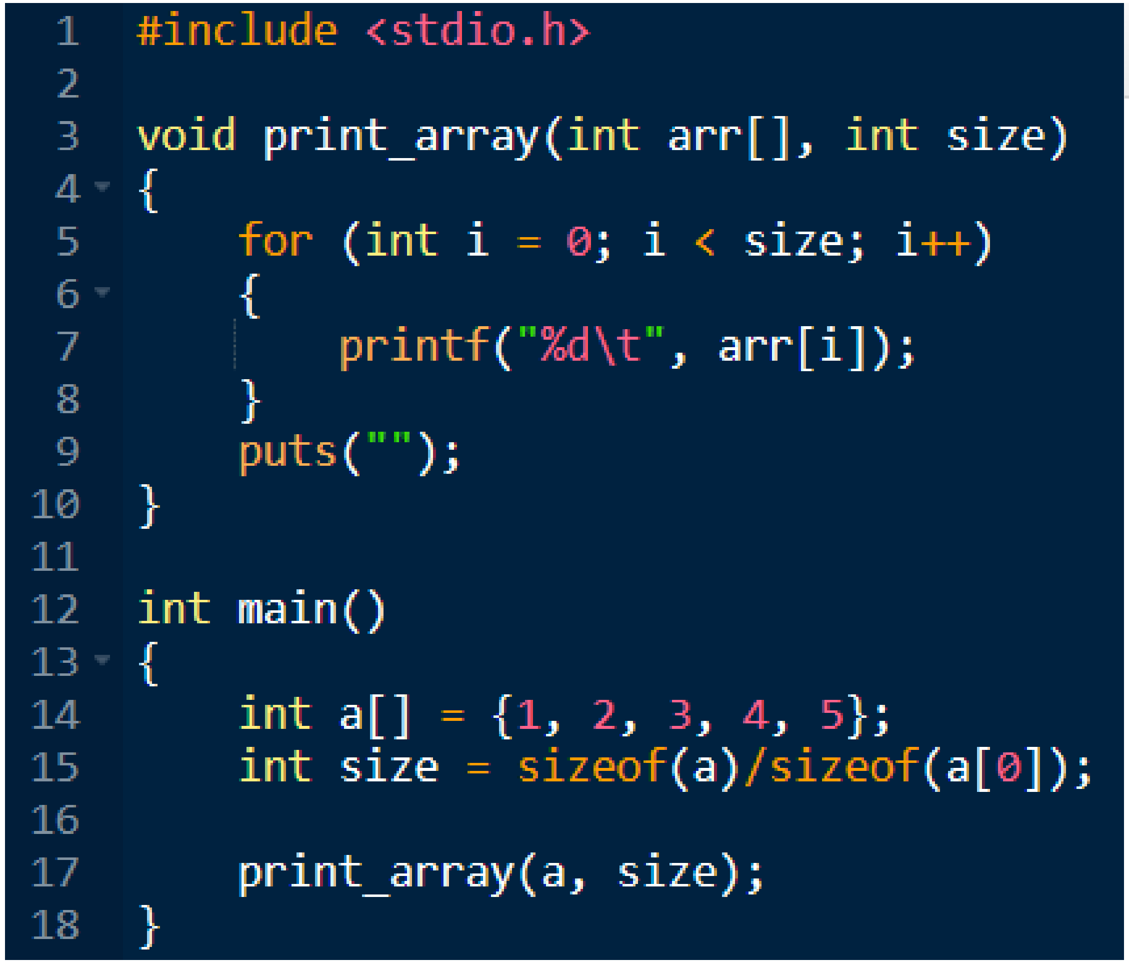
```c
#include <stdio.h>

void print_array(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\t", arr[i]);
    }
    puts("");
}

int main()
{
    int a[] = {1, 2, 3, 4, 5};
    int size = sizeof(a)/sizeof(a[0]);

    print_array(a, size);
}
```



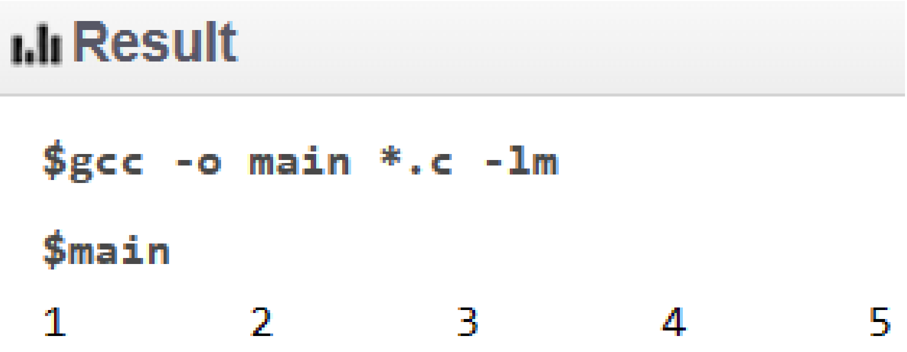
Result

```
$gcc -o main *.c -lm

$main

1       2       3       4       5
```

2. FUNCTION

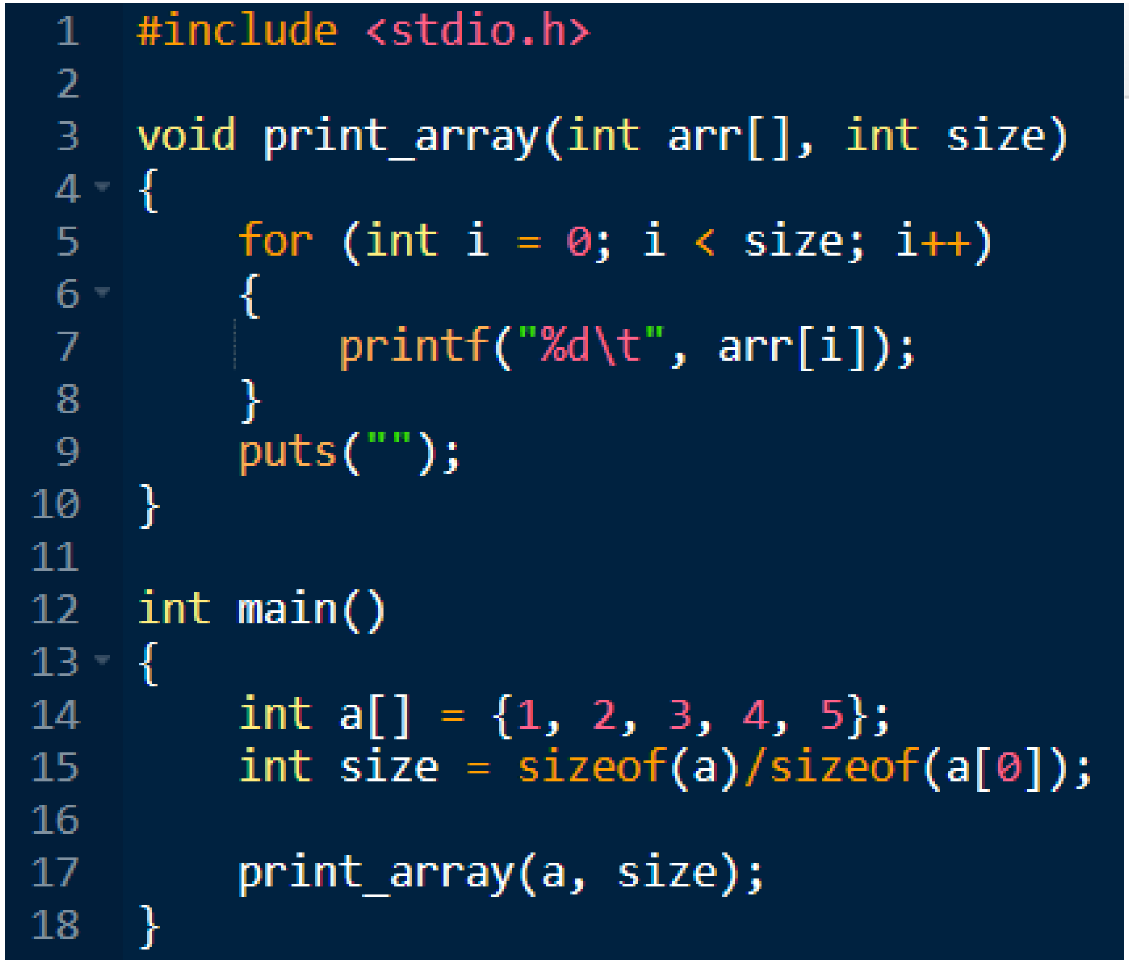


```c
#include <stdio.h>

void print_array(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\t", arr[i]);
    }
    puts("");
}

int main()
{
    int a[] = {1, 2, 3, 4, 5};
    int size = sizeof(a)/sizeof(a[0]);

    print_array(a, size);
}
```

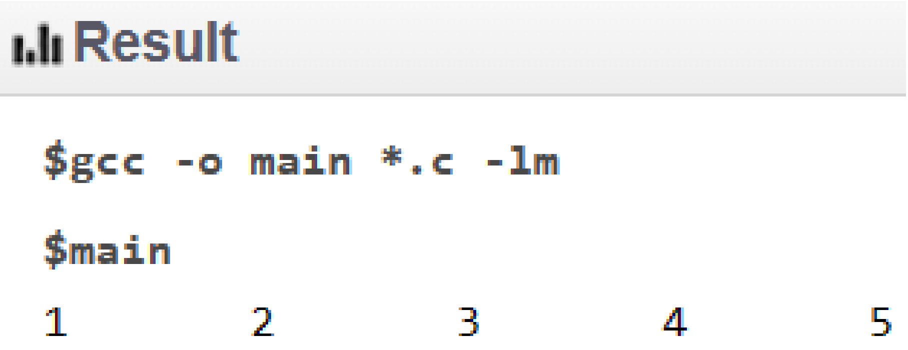


```
$gcc -o main *.c -lm

$main

1       2       3       4       5
```

2. FUNCTION

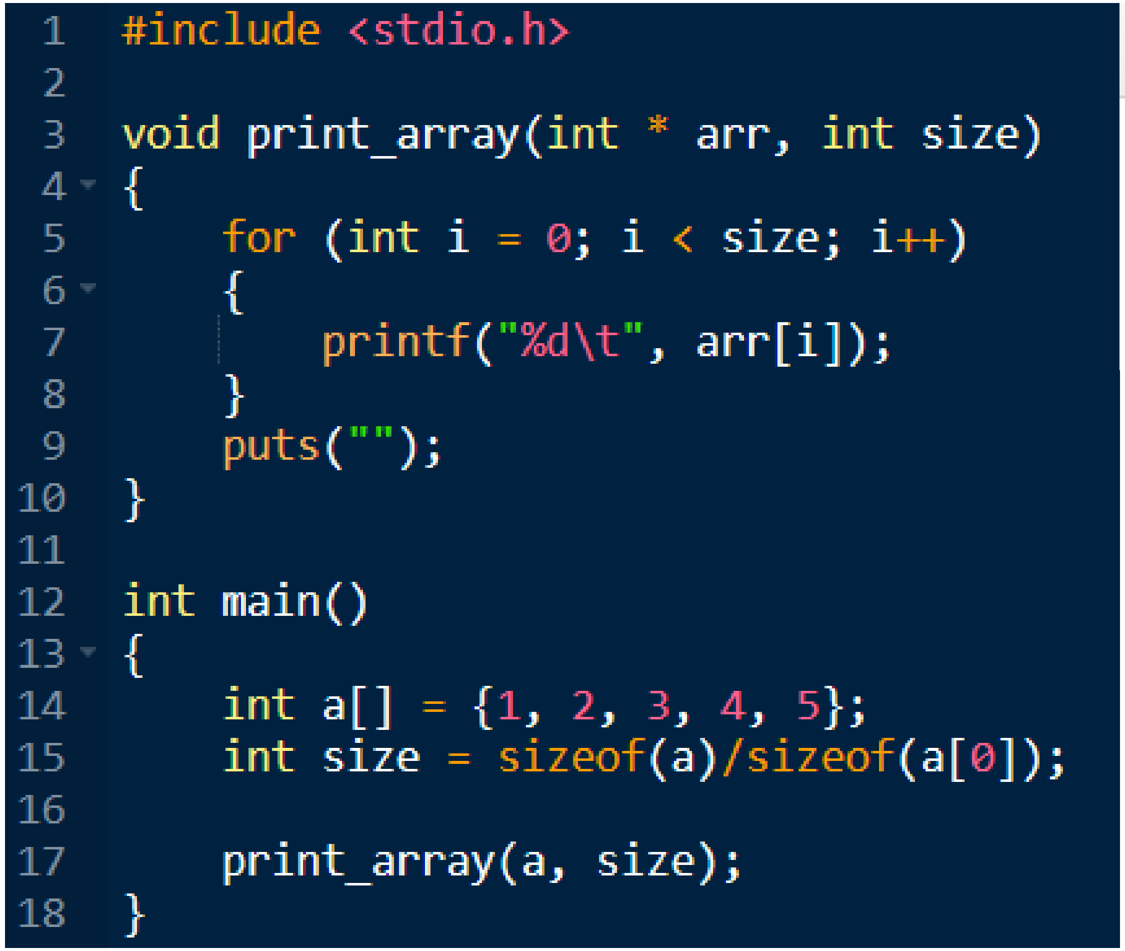


```c
#include <stdio.h>

void print_array(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\t", arr[i]);
    }
    puts("");
}

int main()
{
    int a[] = {1, 2, 3, 4, 5};
    int size = sizeof(a)/sizeof(a[0]);

    print_array(a, size);
}
```

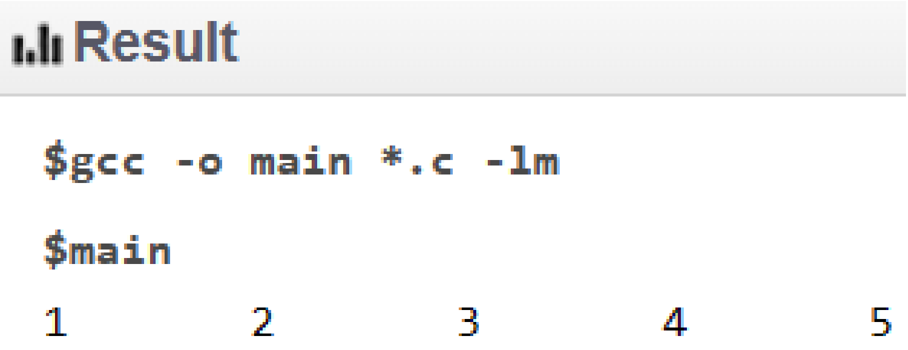


```
$gcc -o main *.c -lm

$main

1       2       3       4       5
```

2. FUNCTION

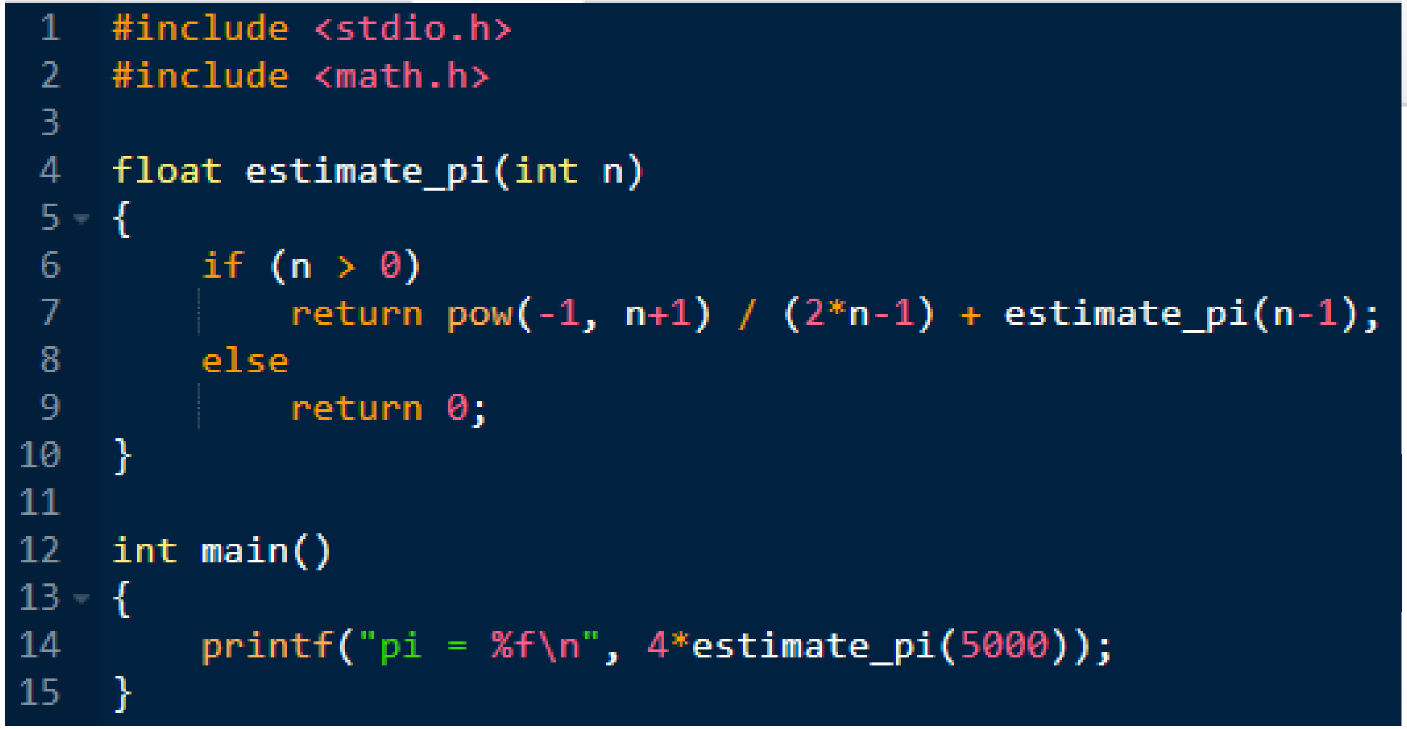


```c
#include <stdio.h>

void print_array(int * arr, int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%d\t", arr[i]);
    }
    puts("");
}

int main()
{
    int a[] = {1, 2, 3, 4, 5};
    int size = sizeof(a)/sizeof(a[0]);

    print_array(a, size);
}
```

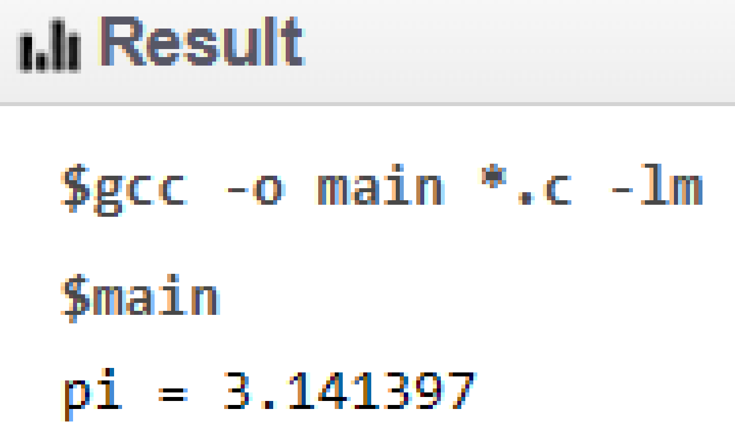


```
$gcc -o main *.c -lm

$main
1       2       3       4       5
```

3. EXERCISES



```
#include <stdio.h>
#include <math.h>

float estimate_pi(int n)
{
    if (n > 0)
        return pow(-1, n+1) / (2*n-1) + estimate_pi(n-1);
    else
        return 0;
}

int main()
{
    printf("pi = %f\n", 4*estimate_pi(5000));
}
```



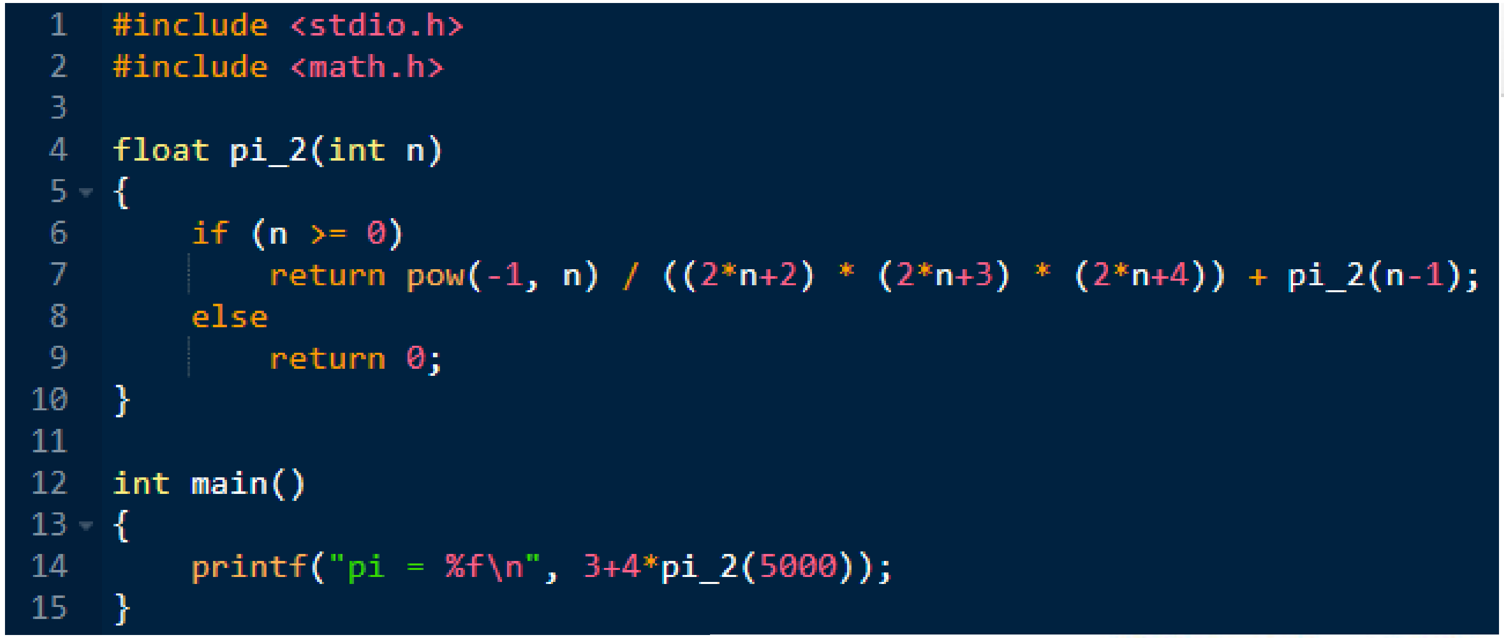
Result

```
$gcc -o main *.c -lm

$main

pi = 3.141397
```

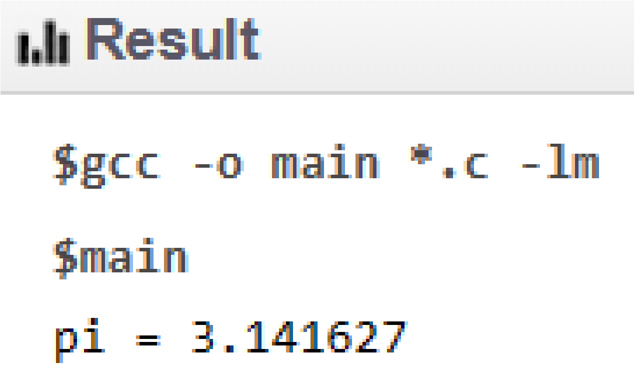
Gregory-Leibniz Series

$$PI \approx 4 \sum_{i=1}^{n} \frac{(-1)^{i+1}}{2i-1}$$

3. EXERCISES



```c
#include <stdio.h>
#include <math.h>

float pi_2(int n)
{
    if (n >= 0)
        return pow(-1, n) / ((2*n+2) * (2*n+3) * (2*n+4)) + pi_2(n-1);
    else
        return 0;
}

int main()
{
    printf("pi = %f\n", 3+4*pi_2(5000));
}
```

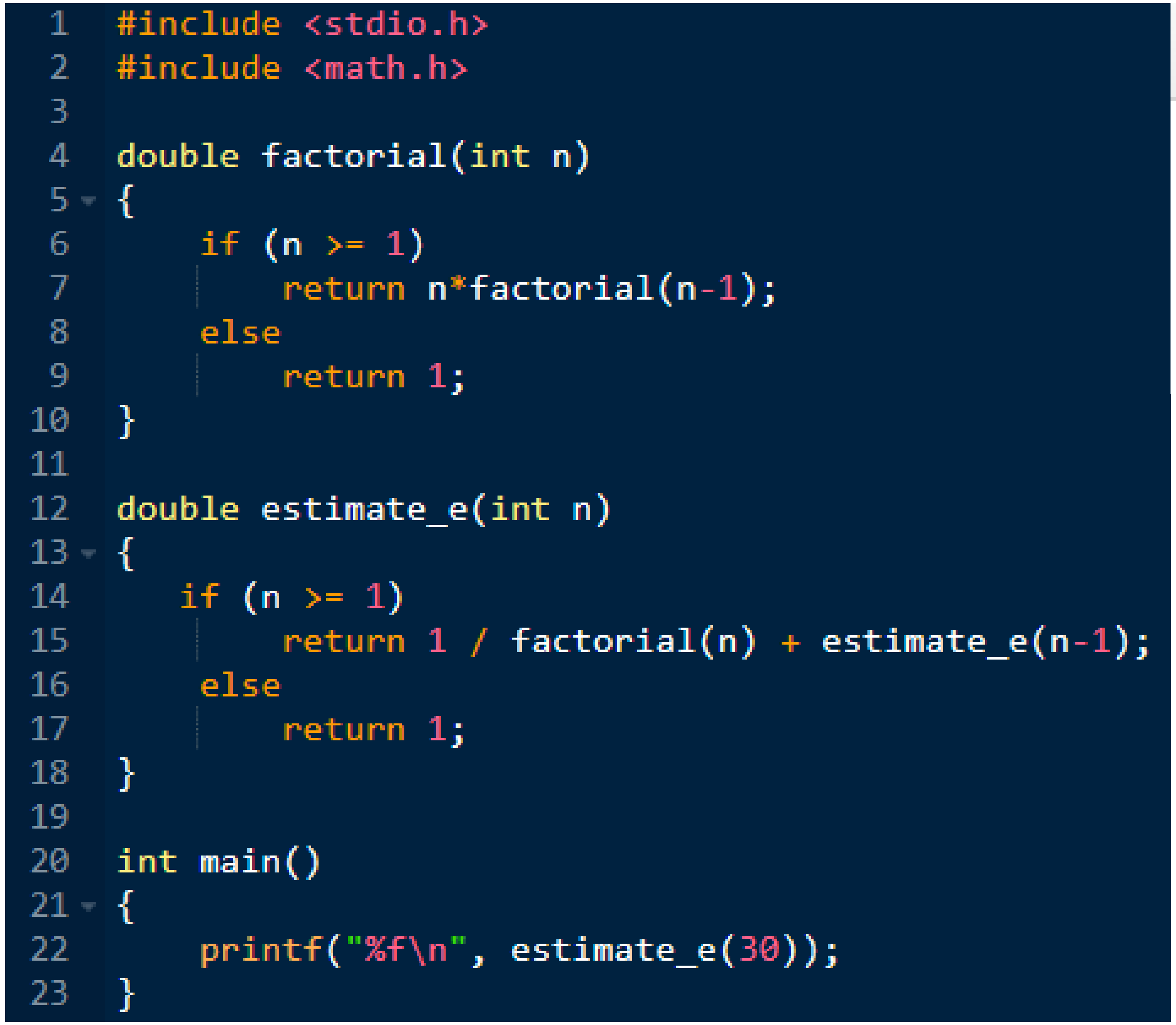


```
$gcc -o main *.c -lm

$main

pi = 3.141627
```

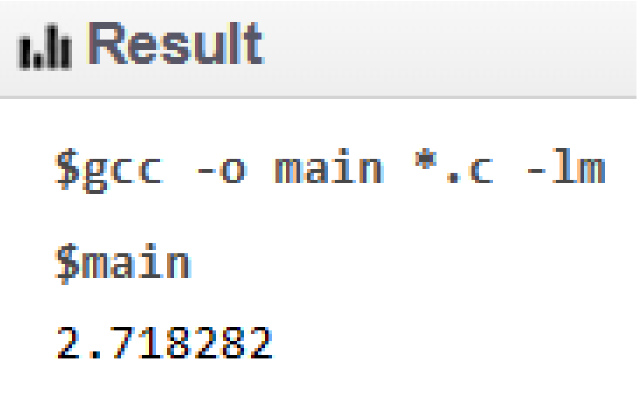
Nilakantha Series

$$PI \approx 3 + 4 \sum_{i=0}^{n} \frac{(-1)^i}{(2i+2)(2i+3)(2i+4)}$$

3. EXERCISES



```c
#include <stdio.h>
#include <math.h>

double factorial(int n)
{
    if (n >= 1)
        return n*factorial(n-1);
    else
        return 1;
}

double estimate_e(int n)
{
   if (n >= 1)
        return 1 / factorial(n) + estimate_e(n-1);
    else
        return 1;
}

int main()
{
    printf("%f\n", estimate_e(30));
}
```

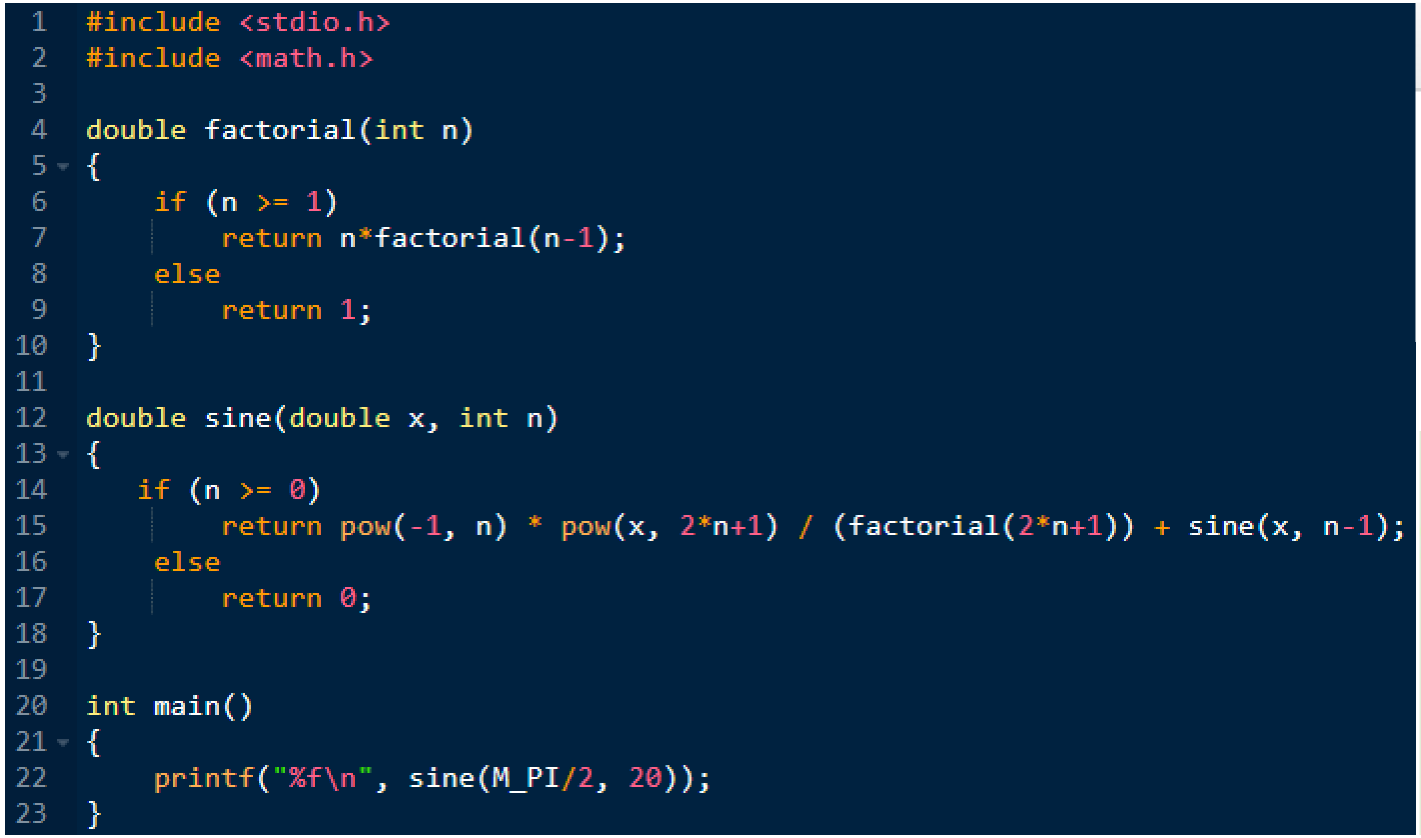


Result

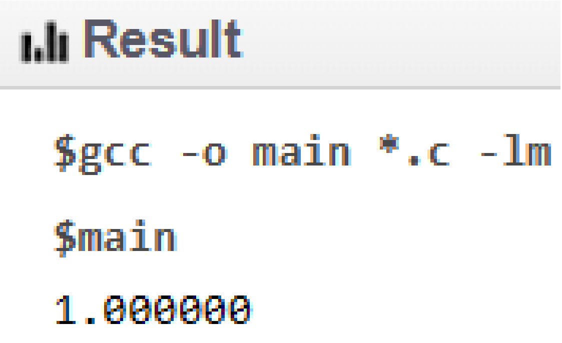
```
$gcc -o main *.c -lm

$main

2.718282
```

$$e \approx 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!}$$

3. EXERCISES



```c
#include <stdio.h>
#include <math.h>

double factorial(int n)
{
    if (n >= 1)
        return n*factorial(n-1);
    else
        return 1;
}

double sine(double x, int n)
{
    if (n >= 0)
        return pow(-1, n) * pow(x, 2*n+1) / (factorial(2*n+1)) + sine(x, n-1);
    else
        return 0;
}

int main()
{
    printf("%f\n", sine(M_PI/2, 20));
}
```



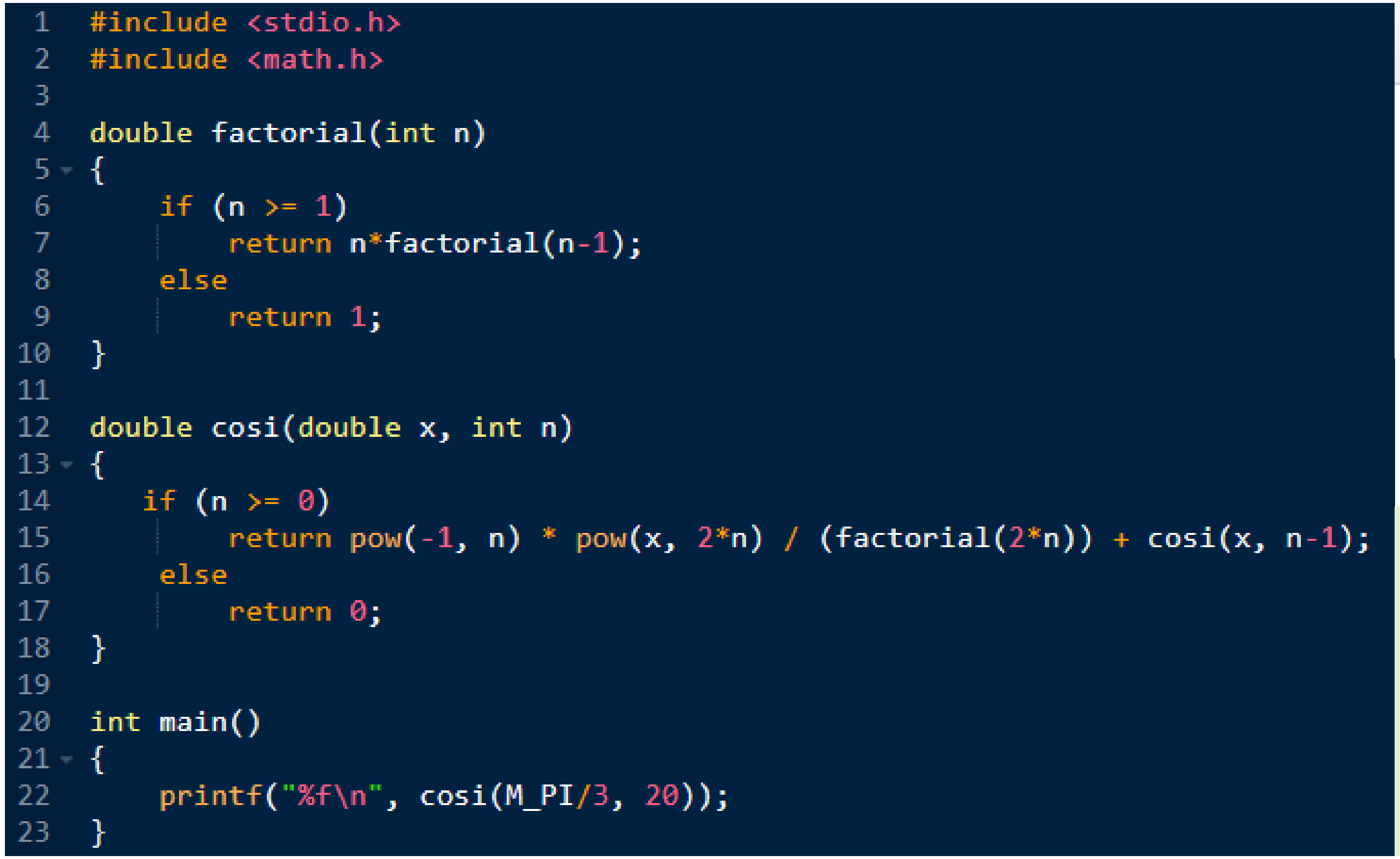
Result

```
$gcc -o main *.c -lm

$main

1.000000
```

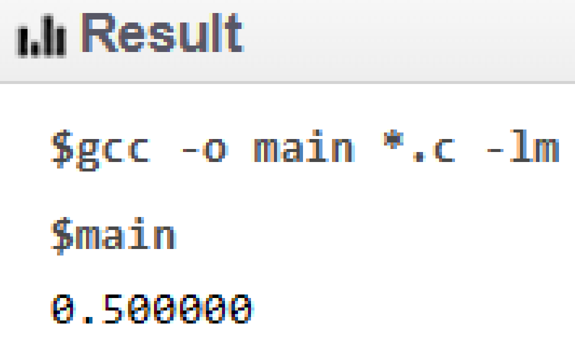
$$sin(x) \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{(2n+1)}}{(2n+1)!}$$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

3. EXERCISES



```c
#include <stdio.h>
#include <math.h>

double factorial(int n)
{
    if (n >= 1)
        return n*factorial(n-1);
    else
        return 1;
}

double cosi(double x, int n)
{
   if (n >= 0)
        return pow(-1, n) * pow(x, 2*n) / (factorial(2*n)) + cosi(x, n-1);
    else
        return 0;
}

int main()
{
    printf("%f\n", cosi(M_PI/3, 20));
}
```



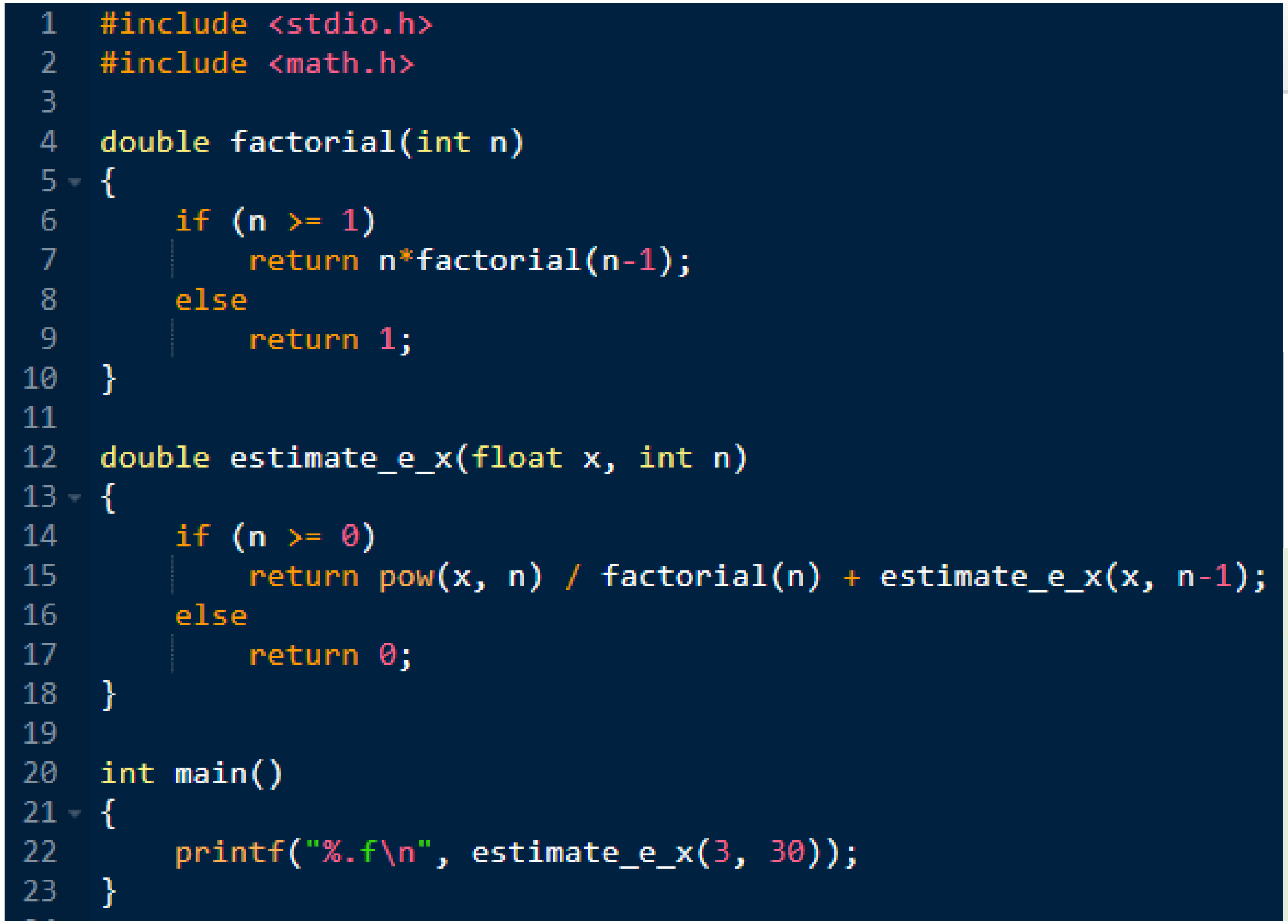
Result

```
$gcc -o main *.c -lm

$main

0.500000
```

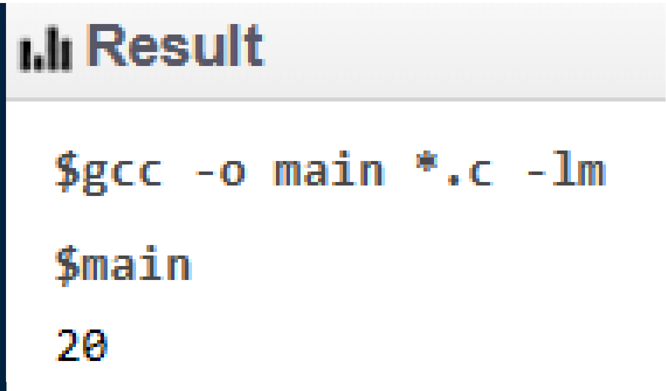
$$cos(x) \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

$$= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \cdots$$

3. EXERCISES



```c
#include <stdio.h>
#include <math.h>

double factorial(int n)
{
    if (n >= 1)
        return n*factorial(n-1);
    else
        return 1;
}

double estimate_e_x(float x, int n)
{
    if (n >= 0)
        return pow(x, n) / factorial(n) + estimate_e_x(x, n-1);
    else
        return 0;
}

int main()
{
    printf("%.f\n", estimate_e_x(3, 30));
}
```



```
$gcc -o main *.c -lm

$main

20
```

$$e^x \approx \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$= 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$