Could you guess what is the output of the following code?

```c
#include <stdio.h>

void function1(void)
{
    int a = 10;
    printf("The value of a in function1 is %d\n", a);
    a = 20;
    return;
}

int main()
{
    int a = -10;
    function1();
    printf("The value of a in main is %d\n", a);
    a = -20;
    function1();
    printf("The value of a in main is %d\n", a);
    return 0;
}
```
Example

What is the output of the following code?

```
[C++] $gcc test.cc -o test.exe
[C++] $./test.exe
The value of a in function1 is 10
The value of a in main is -10
The value of a in function1 is 10
The value of a in main is -20
C++ $
```
Example

What is the output of the following code?

```c
#include <stdio.h>

void function1(void)
{
    int a = 10;
    printf("The value of a in function1 is %d\n", a);
    a = 20;
    return;
}

main()
{
    int a = -10;
    function1();
    printf("The value of a in main is %d\n", a);
    a = -20;
    function1();
    printf("The value of a in main is %d\n", a);
    return 0;
}
```

value of a

<table>
<thead>
<tr>
<th>main</th>
<th>function1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>10</td>
</tr>
</tbody>
</table>
Example

What is the output of the following code?

```c
#include <stdio.h>

void function1(void)
{
    int a = 10;
    printf("The value of a in function1 is %d\n", a);
    a = 20;
    return;
}

int main()
{
    int a = -10;
    function1();
    printf("The value of a in main is %d\n", a);
    a = -20;
    function1();
    printf("The value of a in main is %d\n", a);
    return 0;
}
```

<table>
<thead>
<tr>
<th>main</th>
<th>function1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>20</td>
</tr>
</tbody>
</table>
Example

What is the output of the following code?

```
#include <stdio.h>

void function1(void)
{
    int a = 10;
    printf("The value of a in function1 is %d\n", a);
    a = 20;
    return;
}

int main()
{
    int a = -10;
    function1();
    printf("The value of a in main is %d\n", a);
    a = -20;
    function1();
    printf("The value of a in main is %d\n", a);
    return 0;
}
```

- Value of `a` in `main`:
  - `-10`

- Value of `a` in `function1`:
  - `gone!`
Example

What is the output of the following code?

```c
#include <stdio.h>

void function1(void)
{
    int a = 10;
    printf("The value of a in function1 is %d\n", a);
    a = 20;
    return;
}

int main()
{
    int a = -10;
    function1();
    printf("The value of a in main is %d\n", a);
    a = -20;
    function1();
    printf("The value of a in main is %d\n", a);
    return 0;
}
```

value of a

<table>
<thead>
<tr>
<th>main</th>
<th>function1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>gone!</td>
</tr>
</tbody>
</table>
Example

What is the output of the following code?

```c
#include <stdio.h>

void function1(void)
{
    int a = 10;
    printf("The value of a in function1 is %d\n", a);
    a = 20;
    return;
}

int main()
{
    int a = -10;
    function1();
    printf("The value of a in main is %d\n", a);
    a = -20;
    function1();
    printf("The value of a in main is %d\n", a);
    return 0;
}
```

<table>
<thead>
<tr>
<th>value of a</th>
<th>main</th>
<th>function1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Places for Declaring Variables

```c
int a;

int functionA(int c) {
    int b;
}

int functionB(int c) {
    int b;
}

int main() {
    int a;
    {
        int a;
        {
            int a;
            ...
        }
    }
}

int a;

int functionC(int c) {
    int b;
}

int functionD(int c) {
    int b;
}
```

file1.c

```c
int a;
int b;

int functionC(int c) {
    int b;
}
```

file2.c

```c
int a;
int b;

int functionD(int c) {
    int b;
}
```

file3.c
Places for Declaring Variables

int a;

int functionA(int c)
{
    int b;
}

int functionB(int c)
{
    int b;
}

int main()
{
    int a;
    {
        int a;
        {
            int a;
            ...
        }
    }
}

int a;
int b;

int functionC(int c)
{
    int b;
}

int functionD(int c)
{
    int b;
}
Places for Declaring Variables

**file1.c**

```c
int a;

int functionA(int c)
{
    int b;  // 2
}

int functionB(int c)
{
    int b;  // 2
}

int main()
{
    int a;  // 2
    {
        int a;
        {
            int a;  // 2
            ...
        }
    }
}
```

**file2.c**

```c
int a;
int b;

int functionC(int c)
{
    int b;  // 2
}
```

**file3.c**

```c
int a;

int functionD(int c)
{
    int b;  // 2
}
```
Places for Declaring Variables

```c
int main()
{
    int a;
    {  // 3 places
        int a;  // inside a function
        int a;  // inside a loop
        int a;  // inside a function
    }
}
for(int i=0; i < 10; i++)
{
    int a = 10;
}

int functionA(int c)
{
    int b;
}

int functionB(int c)
{
    int b;
}
```

Places for Declaring Variables

```c
int a;

int functionA(int a)
{
    int b;
}

int functionB(int a)
{
    int b;
}

int main()
{
    int a;
    ...
}

int functionC(int a)
{
    int b;
}

int functionD(int a)
{
    int b;
}
```

Which is which?
Organization

• Scope of variables [applies to most of programming languages]:

  – **Visibility**: where does a variable can be "seen" and used?
  – **Duration (lifetime)**: when does a variable “exist”?

• Scope rules
Scope of Variables
The scope of an identifier is the region of the program in which the identifier can be referenced. Some identifiers can be referenced throughout the program. Others can be referenced from only portions of a program.
• The **scope** of an identifier is the **region of the program** in which the identifier can be referenced.
Local Variables

Variables that are declared inside a function or block are called local variables.

1. A variable declared in the body of a function is **local** to the function

2. A variable declared inside an enclosing block

3. Parameters of a function
Local Variables: Function Variables

• The scope, that is the visibility and validity of the variables inside a function are local (function scope).
Local Variables: Function Scope

• The scope, that is the visibility and validity of the variables inside a function are local (function scope).

```c
#include <stdio.h>

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

int main()
{
    int num1 = 4, num2 = 7;
    int c = sumOfNumbers(num1, num2);
    printf("The sum of %d and %d is %d.\n", num1, num2, sum);
    return c;
}
```
Local Variables: Function Variables

• The scope, that is the visibility and validity of the variables inside a function are local.

The variable **sum** is a local variable! not known outside the function sumOfNumbers!
Function Variables

- Variables with **function scope** are the variables apply only to that particular function, are distinct from other variables of the same name (if any) declared elsewhere in the program outside the function.

- When a function finishes and returns, the value of local variables are gone!
Local Variables: Function Parameters

• The arguments of a function are local to the function and are not visible outside!
Function Parameters

• The arguments of a function are **local** to the function and are not visible outside!

```
#include <stdio.h>

int maxOfNumbers(int a, int b)
{
    return a>b? a: b;
}

int main()
{
    int num1 = 4, num2 = 7;
    int c = maxOfNumbers(num1, num2);
    printf("The max of %d and %d is %d.\n", a, b, c);
    return c;
}
```
Function Parameters

• The arguments of a function are **local** to the function.
Function Parameters

• The arguments of a function are **local** to the function
Local Variables: Block Variables

• A block forms a new scope in which we can define variables (block scope)

• Block: a compound statement

  { declarations statements}

Note: function scope is a special case of block scope
Block Variables

- A block forms a new scope in which we can define variables (block scope)

- Block: a compound statement
  
  ```c
  { declarations statements }
  ```

```c
for (int i=0; i<3; i++)
{
    printf("i is %d\n", i);
}
printf("Now i is: %d\n", i);
```

Wrong!
Block Scope

- A block forms a new scope in which we can define variables (block scope)

```c
#include <stdio.h>

int main()
{
    {
        int a = 3;
    }
    {
        printf("The value of a is %d:\n", a);
    }
    return 0;
}
```
Block Scope

- A block forms a new scope in which we can define variables (block scope)

```c
int main()
{
    {
        a
    }
    {
    }
}
```

Error message:
```
block.cc:11:37: error: use of undeclared identifier 'a'
       printf("The value of a is %d:\n", a);
```

I have no idea what `a` is?
Local Variables: Pros and Cons

- simply created when they are needed (inside a for loop)
- only available from within the routine in which they were created
- memory requirement is less
- easy to debug
Global Variables

Variables declared outside the body of any function, usually on top of the program

- File scope: visible from declaration until end of enclosing file
Global Variables

```c
#include <stdio.h>

int sum;

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

int main()
{
    int num1 = 4, num2 = 7;
    sumOfNumbers(num1, num2);
    printf("The sum of %d and %d is %d.\n", num1, num2, sum);
    return 0;
}
```

```
gcc sumlocal.cc -o sumlocal.exe
./sumlocal.exe
The sum of 4 and 7 is 11.
```
Global Variables: Pros and Cons

- Global variable
  - declared very early stage
  - available at all times from anywhere
  - created at the start of the program, and lasts until the end, it is taking up lots of memory
  - difficult to debug
Global Variables: Pros and Cons

• **Convenient** way for functions to share variables (+ multiple outputs)

• **Maintenance issues:**
  – If type changes, we need to check every function that uses it
  – If assigned wrong value: may be difficult to locate where
  – Functions that rely on external are hard to reuse
Scope Rules

- Scope rules: used for **name resolution**
  - i.e., we can use the same name for different variables (local and global variables), then how to determine "which is which"

The value of local variable will take preference.

The parameters of a function are treated as local variables and they take precedence over global variables
Scope Rules

Find the closest definition of the variable and use it!
Scope Rules

Find the closest definition of the variable and use it!

```c
int a = 1;

int functionA(int a)
{
    print a;
}

int functionB(int d)
{
    print a;
}

int main()
{
    print a;
    {
        int a = 2;
        print a;
    }
}
```
Exercise (I): Scope Rules

```c
#include <stdio.h>

int num = 0;

int main()
{
    printf("1: The value of num is %d:\n", num);
    {
        int num = 1;
        printf("2: The value of num is %d:\n", num);
        {
            int num = 2;
            printf("3: The value of num is %d:\n", num);
            {
                int num = 3;
                printf("4: The value of num is %d:\n", num);
            }
        }
    }
    printf("5: The value of num is %d:\n", num);
    printf("6: The value of num is %d:\n", num);
}
    printf("7: The value of num is %d:\n", num);
}
return 0;
```
Scope Rules

```c
#include <stdio.h>

int num = 0;

int main()
{
    printf("1: The value of num is %d:\n", num);
    {
        int num = 1;
        printf("2: The value of num is %d:\n", num);
        {
            int num = 2;
            printf("3: The value of num is %d:\n", num);
            {
                int num = 3;
                printf("4: The value of num is %d:\n", num);
            }
            printf("5: The value of num is %d:\n", num);
        }
        printf("6: The value of num is %d:\n", num);
    }
    printf("7: The value of num is %d:\n", num);
    return 0;
}
```
Scope Rules

What is the output of the following code?

```c
#include <stdio.h>

float a = 17;       // global constant
def b;              // global variable
def c;              // global variable

void func(float c)   // prevents access to global c
{
def b;              // prevent access to global b
    b = 2.3;        // assignment to local b
    printf("a = %f\n", a); // output global a (17)
    printf("b = %f\n", b); // output local b (2.3
    printf("c = %f\n", c); // output local c (42.8)
}

int main()
{
b = 4;                    // assignment to global b
c = 6;                    // assignment to global c
func(42.8);
return 0;
```
Scope Rules

What is the output of the following code?

```c
#include <stdio.h>

float a = 17;       // global constant
float b;            // global variable
float c;            // global variable

void func(float c)  // prevents access to global c
{
    float b;        // prevent access to global b
    b = 2.3;        // assignment to local b
    printf("a = %f\n", a); // output global a (17)
    printf("b = %f\n", b); // output local b (2.3)
    printf("c = %f\n", c); // output local c (42.8)
}

int main()
{
    b = 4;          // assignment to global b
    c = 6;          // assignment to global c
    func(42.8);     // assignment to global c
    return 0;
}
```

<table>
<thead>
<tr>
<th>variable</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>17.0000</td>
</tr>
<tr>
<td>b</td>
<td>2.3000</td>
</tr>
<tr>
<td>c</td>
<td>42.8</td>
</tr>
</tbody>
</table>
Scope: Suggestion

• Suggestion: avoid using the same name for different variables at different places of your code, unless the context is very clear for a variable's scope
  – E.g., using the same name for different local/block variables in different functions/compound statements is fine, but using the same name for a global variable and a local variable can make your code confusing