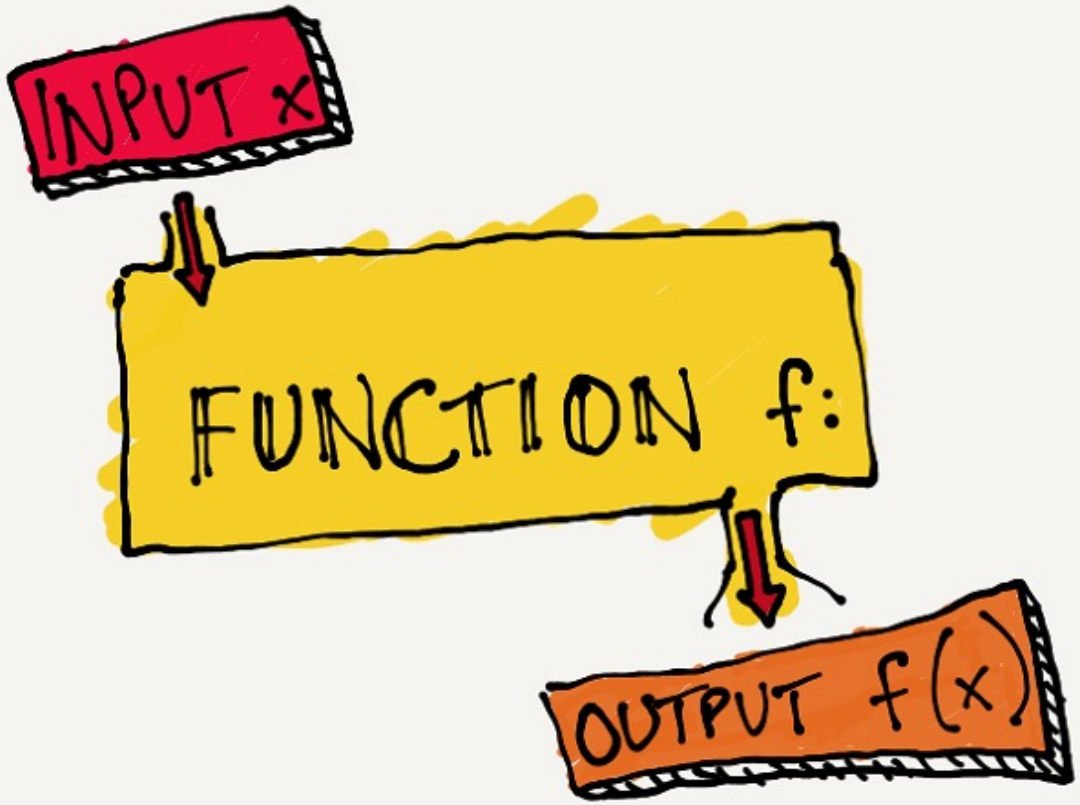
ALGORITHM AND COMPUTATIONAL THINKING 2

WEEK 2 – Functions (part 2)



QUIZ

What will this code print?

```
int processNumber(int x) {
    return x * 2;
}

int main() {
      int result = processNumber(3);
      result += processNumber(result);
      printf("%d\n", result);
      return 0;
}
```

A. 6

B. 12

C. 18

D. 3

ANSWER

What will this code print?

```
int processNumber(int x) {
    return x * 2;
}

int main() {
      int result = processNumber(3);
      result += processNumber(result);
      printf("%d\n", result);
      return 0;
}
```

A. 6

B. 12

C. 18

D. 3

QUIZ

What will this code print?

```
void printSquare(int x) {
    printf("Square is: %d ", x * x);
}

int main() {
    int result = printSquare(5);
    printf(“ - Result: %d\n", result);
    return 0;
}
```

A. Square is: 25

B. Square is: 25 – Result: 25

C. Compilation Error

D. Result: 25

ANSWER

What will this code print?

```
void printSquare(int x) {
    printf("Square is: %d ", x * x);
}

int main() {
    int result = printSquare(5);
    printf(“ - Result: %d\n", result);
    return 0;
}
```

A. Square is: 25

B. Square is: 25 – Result: 25

C. Compilation Error ← You are expected an integer as return, but the function returns nothing

D. Result: 25

QUIZ

What will this code print?

```
int checkValue(int x) {
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
}

int main() {
    printf("%d\n", checkValue(-3));
    return 0;
}
```

A. 1

B. 0

C. Undefined value

D. Compilation error

ANSWER

What will this code print?

```
int checkValue(int x) {
    if (x > 0)
        return 1;
    else if (x == 0)
        return 0;
}

int main() {
    printf("%d\n", checkValue(-3));
    return 0;
}
```

Case x<0 is **NOT handled** here !

A. 1

B. 0

C. Undefined value

D. Compilation error

Some compilers allow it but it leads to **unpredictable output**

Session objectives

- ✓ Distinguish between **local and global** variable scopes.
- ✓ Understand variable **memory size**.
- ✓ Explain how **variables and arrays are passed to functions**.
- ✓ Be able to compute the **number of elements in array** dynamically.

Variable **Scope**

*Let's lean how variable **visibility** and **lifetime** affect program behavior…*

*What is a **block of code** and how variable **are visible** within blocks?*

QUIZ

What will this code print?

```
int main() {
    if (true) {
      int y = 10;
    }

    printf("Y = %d", y);
    return 0;
}
```

A. error: 'y' undeclared

B. 10

C. undefined

D. 0

ANSWER

What will this code print?

```
int main() {
    if (true) {
      int y = 10;
    }

    printf("Y = %d", y);
    return 0;
}
```

The variable Y exists only during this bloc lifetime !

A. error: 'y' undeclared

B. 10

C. undefined

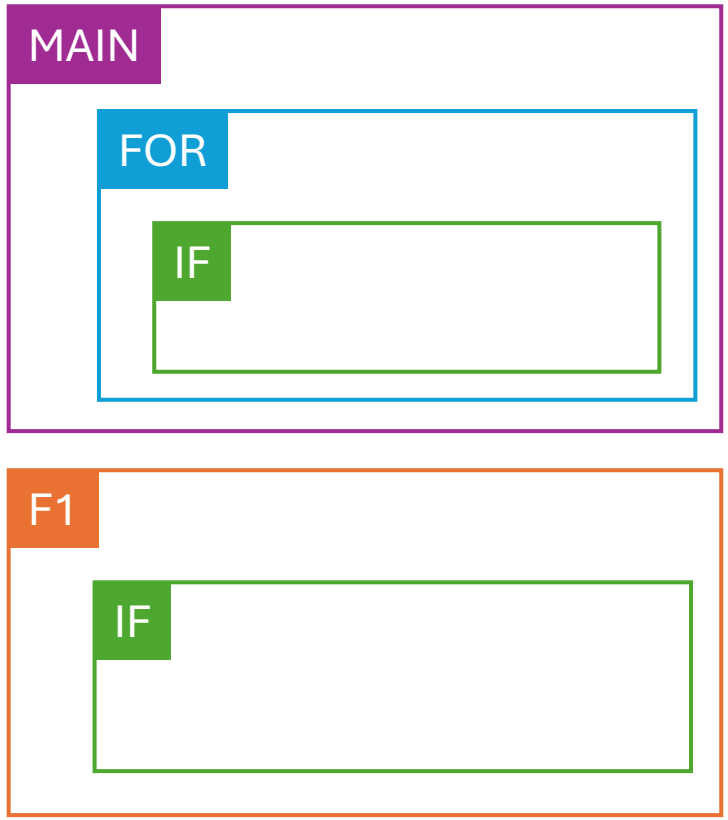
D. 0

EXPLAIN

Block Representation

*We represent each block statement (functions, if-else, loops..) using **blocks***

```
int main() {
   for (…) {
        if (…){
         }
   }
}

void f1(int z) {
        if (…){
         }
}
```



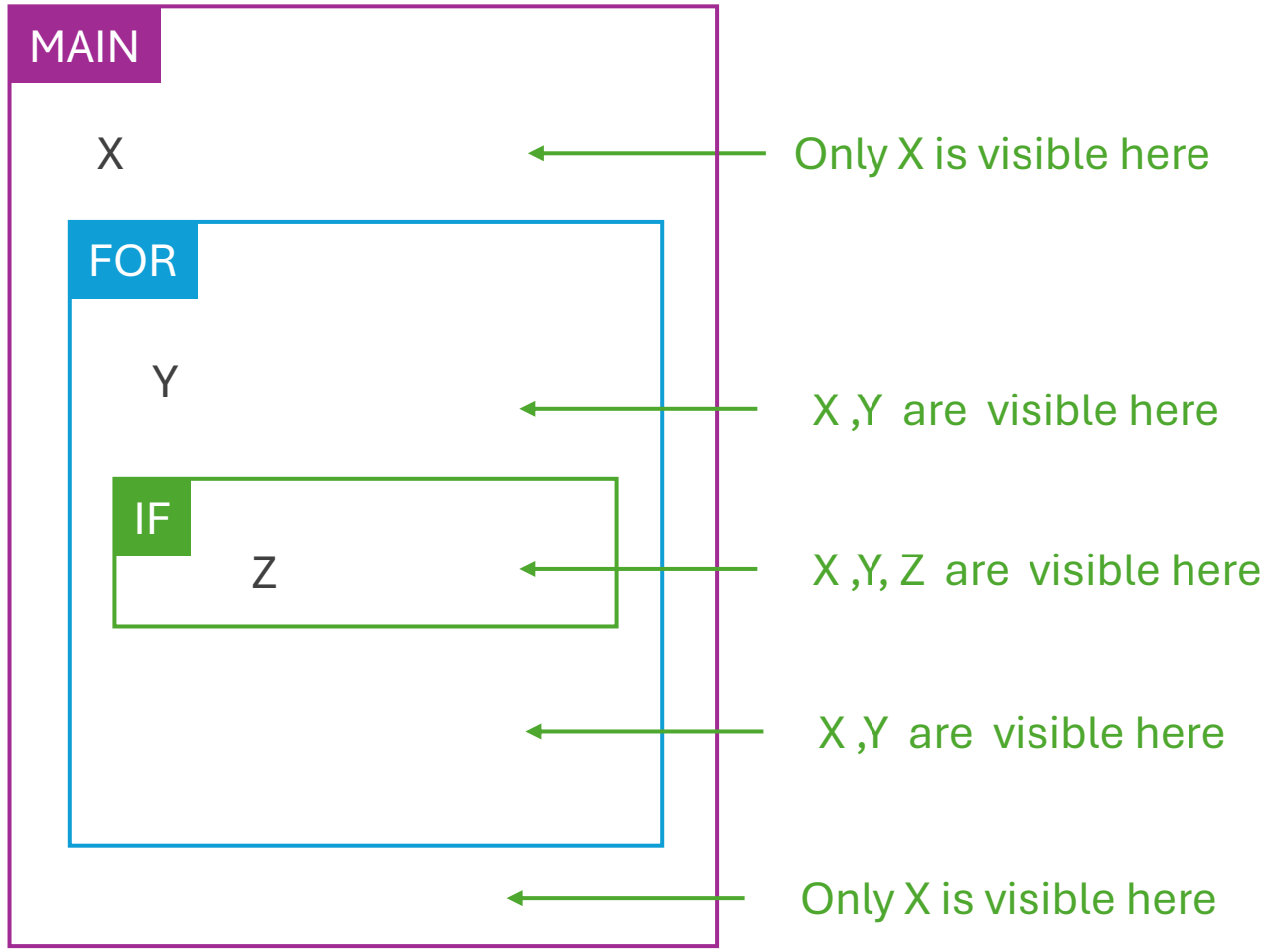
Each block can contain other nested blocks

EXPLAIN

Variable **scope**

*Variables are only accessible **inside the block { }** they are created*

```
int main() {
   int x = 4;

   for (…) {
      int y = 4;

      if (…){
         int z = 4;
       }
   }
}
```



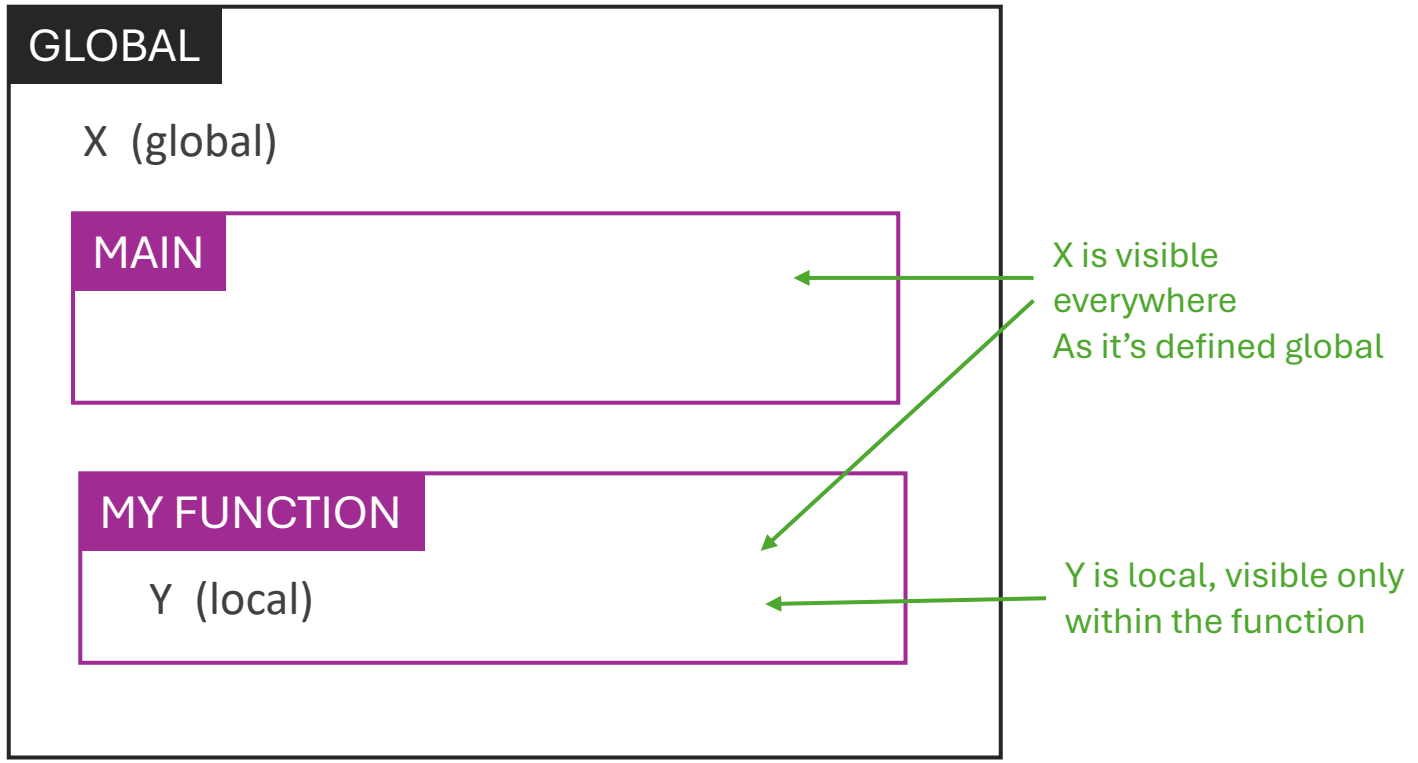
A variable is visible within the block where it is declared and in any nested (child) block beneath it.

EXPLAIN

Global & **Local** variables

- ✓ Variables outside of any block { } are **global** and are available anywhere.
- ✓ Variables inside a block { } are **local** and are visible only within this block.

```
int x = 10;

int main() {
    printf("%d", g);
}

void myFunction() {
    int y = 5
    printf("%d", g);
}
```

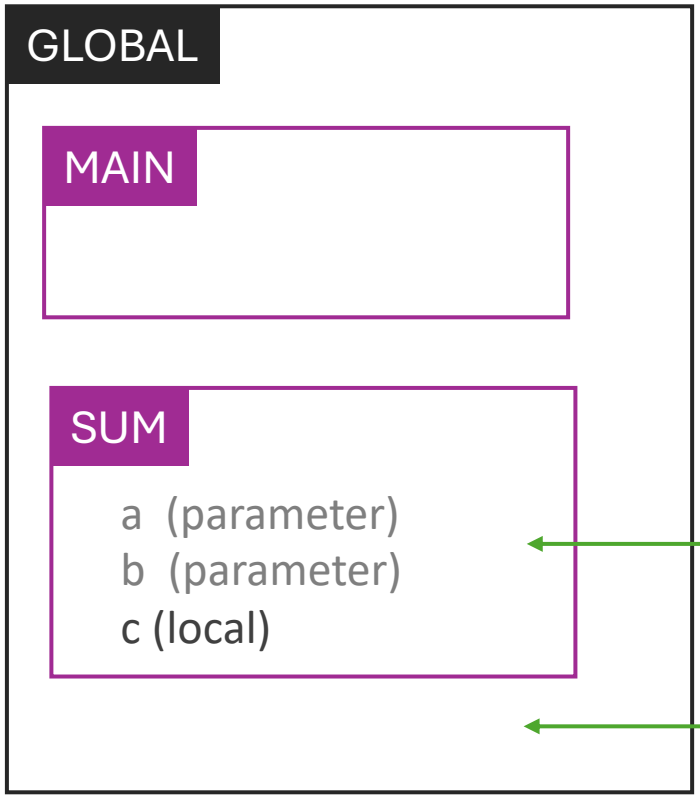


EXPLAIN

Function **parameters visibility**

- ✓ Function parameters are like local variables in the function.
- ✓ they are visible only during the function run.

```
int main() {
    printf("%d", sum(2, 8));
}


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

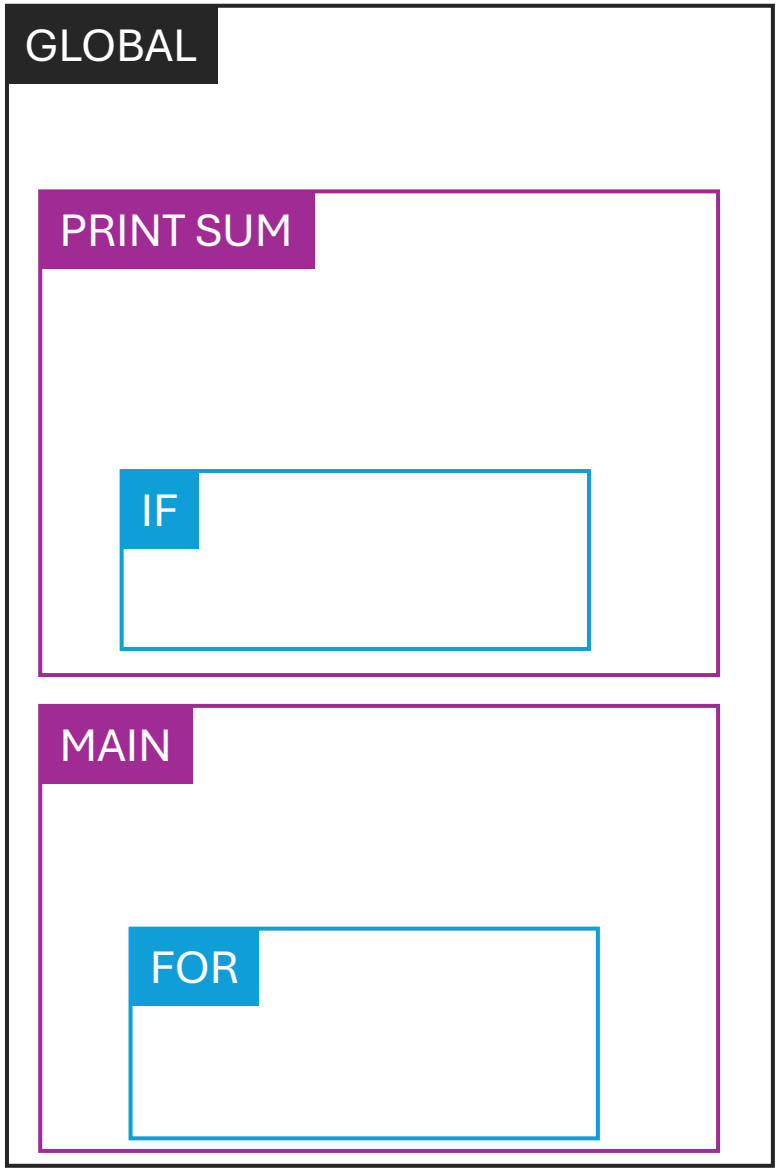


QUIZ

Draw the **Variable Block model** of this code

Identify all variables and where they are declared and complete the bellow block model

```
int g = 100;

void printSum(int a, int b) {
    int sum = a + b;
    if (sum > 10) {
        int bonus = 5;
        sum += bonus;
    }
}

int main() {
    int x = 4;
    int y = 8;
    for (int i = 0; i < 2; i++) {
        int temp = x * i;
        printSum(temp, y);
    }
    return 0;
}
```

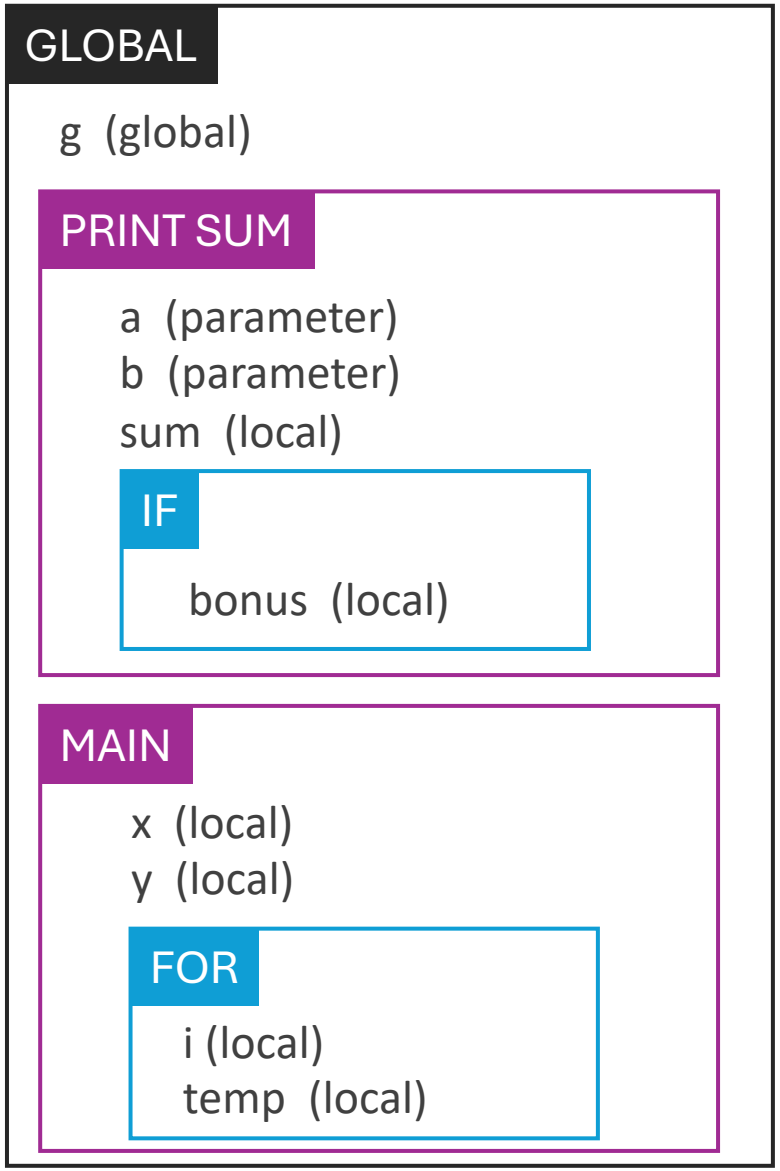


ANSWER

Draw the **Variable Block model** of this code

Identify all variables and where they are declared and complete the bellow block model

```
int g = 100;

void printSum(int a, int b) {
    int sum = a + b;
    if (sum > 10) {
        int bonus = 5;
        sum += bonus;
    }
}

int main() {
    int x = 4;
    int y = 8;
    for (int i = 0; i < 2; i++) {
        int temp = x * i;
        printSum(temp, y);
    }
    return 0;
}
```



QUIZ

Draw the **Variable Block model** of this code

Identify all variables and where they are declared and complete the bellow block model

```
int counter = 0;

void update(int value) {
    int counter = value * 2;
    while (counter > 0) {
         int step = counter % 3;
         counter--;
    }

    if (value > 5) {
        int bonus = 10;
    } else {
        int penalty = -5;
    }
}

int main() {
    int input = 4;
    update(input);

    int result = counter + input;

    return 0;
}
```

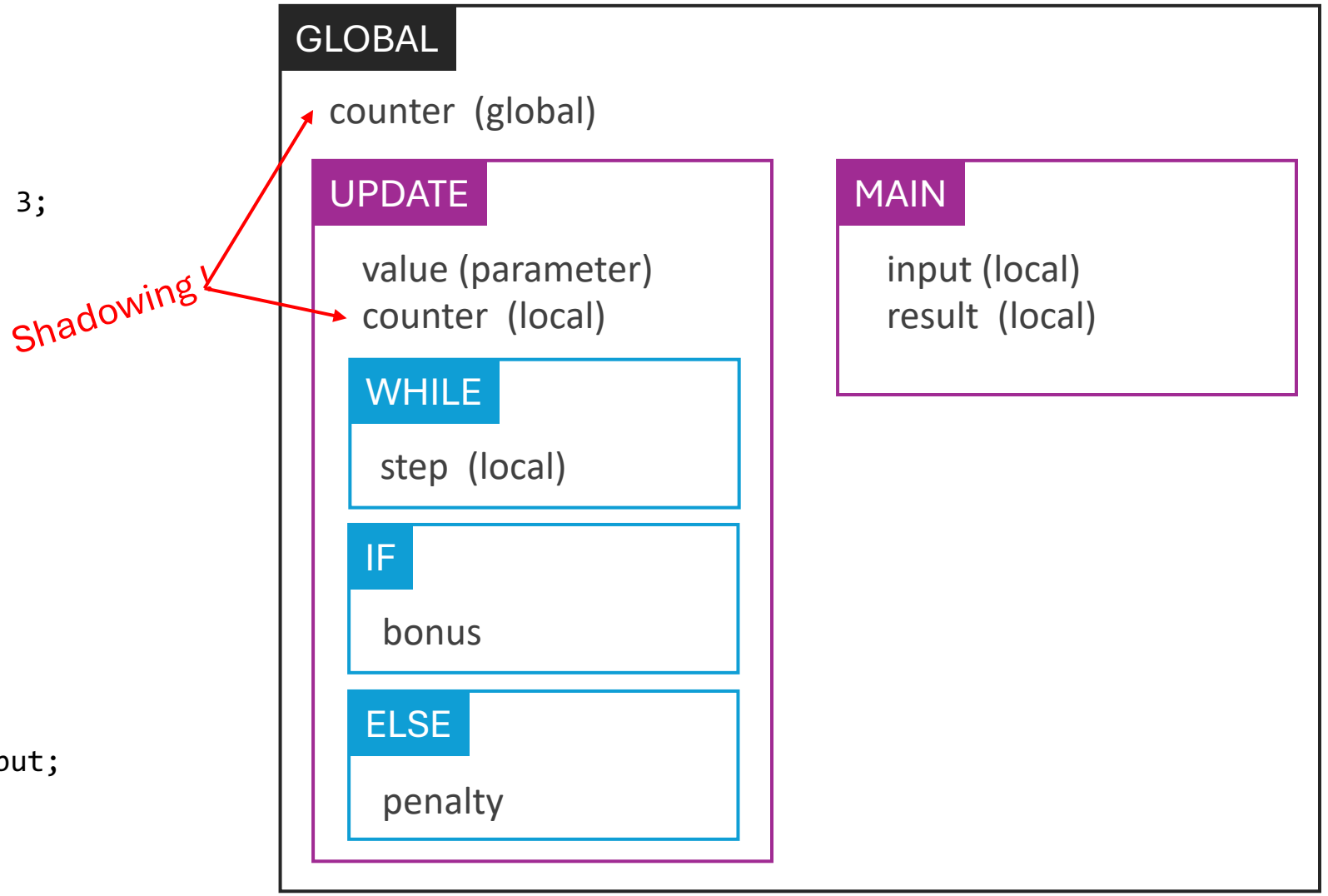
GLOBAL

ANSWER

Draw the **Variable Block model** of this code

Identify all variables and where they are declared and complete the bellow block model

```
int counter = 0;

void update(int value) {
    int counter = value * 2;
    while (counter > 0) {
        int step = counter % 3;
        counter--;
    }

    if (value > 5) {
        int bonus = 10;
    } else {
        int penalty = -5;
    }
}

int main() {
    int input = 4;
    update(input);

    int result = counter + input;

    return 0;
}
```



EXPLAIN

Variable **Shadowing**

Variable shadowing happens when a variable in an **inner scope** has the **same name** as a variable in an **outer scope**.

```
int x = 10;  // global variable

void example() {
    int x = 5;  // shadows the global x
    printf("x in example() = %d\n", x);     // prints 5 (local x)
}

int main() {
    example();
    printf("x in main() = %d\n", x);        // prints 10 (global x)
    return 0;
}
```

*This local x **hides the global one** within the example() function.*

REFLECTION

Memory Concepts

An int variable will occupy **4 bytes** of memory.

```
int x = 10;
```

Q1 – What does a **byte represent**? How many values a byte can contain?

Q2 – How **many values** can be represented with a **variable of type integer** ?

Q3 – What's happen if we define an integer value **outside of the possible range** of values?

ANSWER

Memory Concepts

An int variable will occupy **4 bytes** of memory.

```
int x = 10;
```

Q1 – What does a **byte represent**? How many values a byte can contain?

1 byte = 8 bits (example: 1011 1111 in binary or BF (base 16) in hexadecimal).
With 8 bits we can represent 2^8 values, so 256 values (ranging from 0 to 255 in decimal).

Q2 – How **many values** can be represented with a **variable of type integer** ?

integer = 4 bytes
With an integer, we can represent 2^32 values, so 4,294,967,296 values.

Q3 – What's happen if we define an integer value **outside of the possible range** of values?

It's an **overflow**. It will provide unintended **behavior.**

EXPLAIN

Data type **sizes**

When we declare a variable, the computer allocates some **amount memory** according to the **variable data types**

Type	Description	Size in bytes
Char	Character	1
Int	Integer	4
Float	Floating point	4
Double	Double floating point	8
Bool	Boolean	1

Warning :
It can vary depending on the computer

Understanding their sizes helps you manage memory and write efficient code

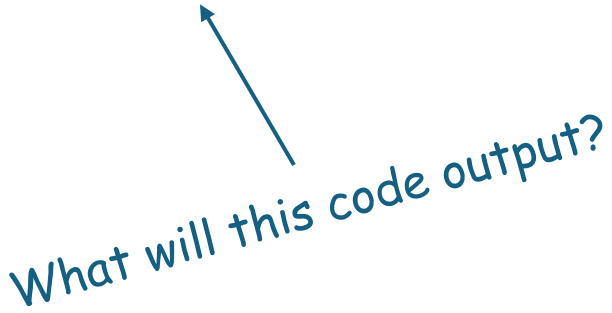
*Note: Use ASCII table (link) to check the values of the **Character**.*

EXPLAIN

Sizeof with types or variables

sizeof is a compile-time operator in C that **returns the number of bytes** used to store a data type or variable in memory.

```
int a = 5;
char c = 'A';
double d = 3.14;

printf("Size of int: %d bytes\n", sizeof(int));
printf("Size of a: %d bytes\n", sizeof(a));
printf("Size of char: %d bytes\n", sizeof(char));
printf("Size of c: %d bytes\n", sizeof(c));
printf("Size of double: %d bytes\n", sizeof(d));
```



ANSWER

Sizeof with types or variables

sizeof is a compile-time operator in C that **returns the number of bytes** used to store a data type or variable in memory.

```
int a = 5;
char c = 'A';
double d = 3.14;

printf("Size of int: %d bytes\n", sizeof(int));
printf("Size of a: %d bytes\n", sizeof(a));
printf("Size of char: %d bytes\n", sizeof(char));
printf("Size of c: %d bytes\n", sizeof(c));
printf("Size of double: %d bytes\n", sizeof(d));
```

```
4 bytes
4 bytes
1 bytes
1 bytes
8 bytes
```

EXPLAIN

Signed int & **Unsigned** int

- ✓ Signed int can represent **both positive** and **negative** values
- ✓ Unsigned int can only represent **positive integer** values.

```
unsigned int a = 5;
```

Type modifier

```
unsigned int x;
x = 4294967295;

printf("%u", x);
```

```
4294967295
```

Unsigned int range is 0.. $2^{32}-1$

```
int x;
x = 4294967295;

printf("%u", x);
```

```
-1
```

Value x is out of range **(overflow)**

Signed int range is -2^{31} ... $2^{31}-1$

QUIZ

Sizeof with **Arrays**

Click Here → MORE INFO

- ✓ Sizeof(myVariable) return the size (in byte) allocated for the variable.
- ✓ Sizeof(myArray) will return the size allocated for the **whole array**.

```
int myNumbers[] = {10, 25, 50, 75, 100};

printf("%lu", sizeof(myNumbers));
```

What will this code output?

ANSWER

Sizeof with **Arrays**

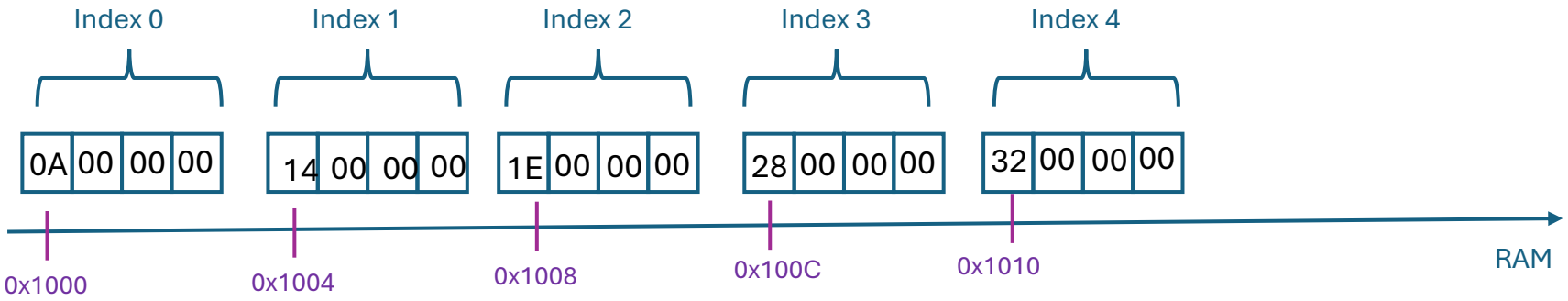
Click Here

MORE INFO

```
int myNumbers[] = {10, 20, 30, 40, 50};

printf("%lu", sizeof(myNumbers));
```

It will return 20:

- Size(int) = 4
- The array has 5 elements of type int: 5 *4 = 20 bytes



EXPLAIN

Click Here

MORE INFO

Array **number of elements**

To compute the number of elements in an array, divide **the total size** of the array by the **size of one element**.

```
int myNumbers[5] = {10, 20, 30, 40, 50};

int count = sizeof(numbers) / sizeof(numbers[0]);
```

20/4 = 5 elements

20

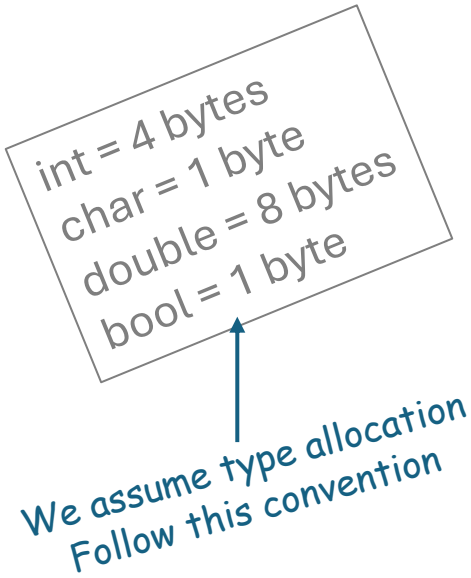
4

QUIZ

How many bytes of memory are allocated in total?

```
double data[10] = {0};
```

A. 10
B. 0
C. 80
D. 40

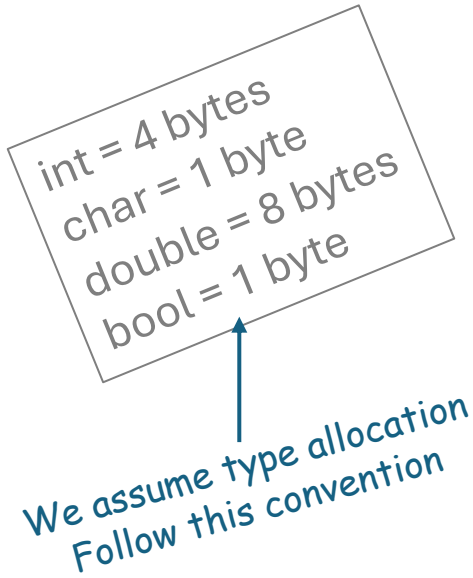


ANSWER

How many bytes of memory are allocated in total?

```
double data[10] = {0};
```

A. 10
B. 0
C. 80 10 * sizeof(double) = 10 *8
D. 40

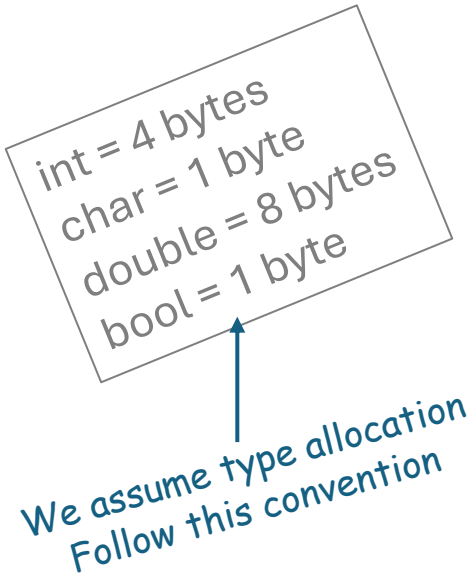


QUIZ

How many bytes of memory are allocated in total?

```
int a;
char b;
double c;
```

A. 13
B. 14
C. 12
D. 15

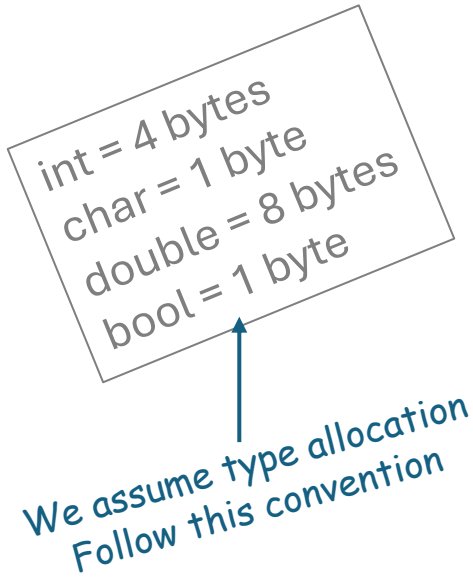


ANSWER

How many bytes of memory are allocated in total?

```
int a;
char b;
double c;
```

A. 13 4 + 1 +8 = 13
B. 14
C. 12
D. 15

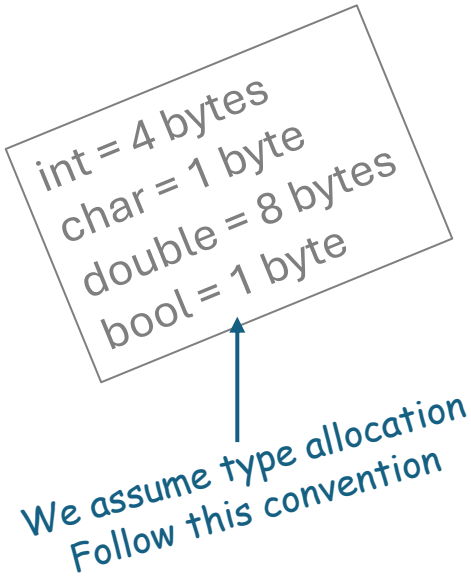


QUIZ

How many bytes of memory are allocated in total?

```
char letters[10];
bool flags[4];
```

A. 12
B. 14
C. 16
D. 10

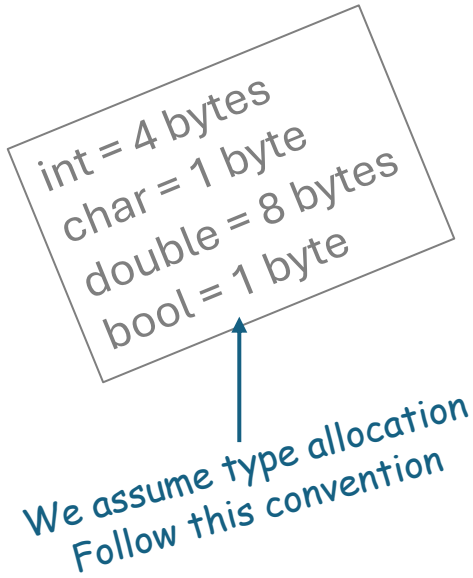


ANSWER

How many bytes of memory are allocated in total?

```
char letters[10];
bool flags[4];
```

A. 12
B. 14 (10 bytes for letters, 4 bytes for flag)
C. 16
D. 10



EXPLAIN

Passing **by value** variables to functions

MORE INFO

Click Here

- ✓ Primitive types (int, bool, double..) are **passed by value**.
- ✓ A **copy of the argument** is passed to the function.
- ✓ The function **works on this copy**, so it will not affect the original variable.

```
void func(int val) {
        // Changing the value
        val = 123;
}

int main() {
        int x = 1;

        // Passing x by value to func()
        func(x);
        printf("%d", x);
        return 0;
}
```

What will this code output?

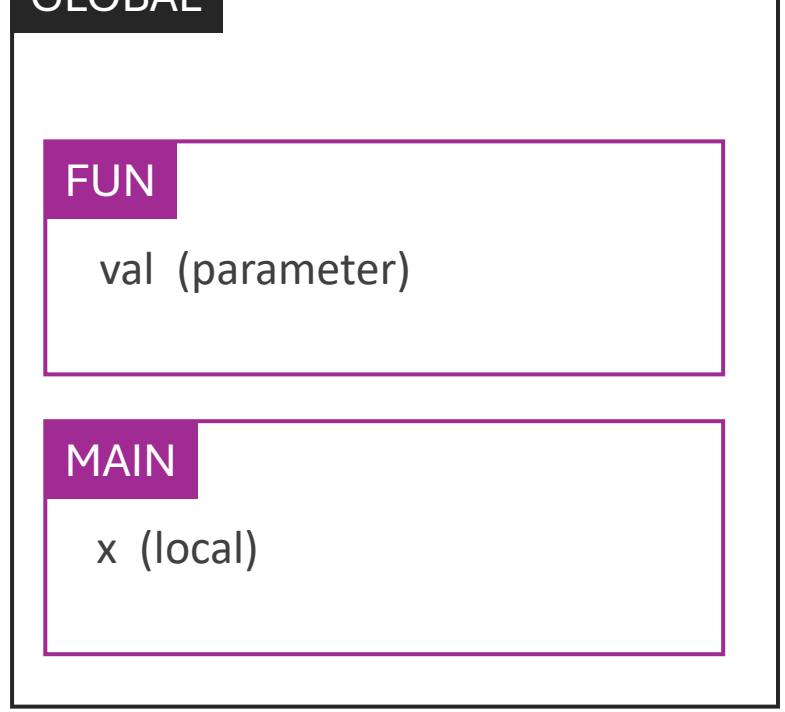
ANSWER

Passing **by value** variables to functions

The variable X is not affected by the function. As we pass to the function a COPY of this variable

```
void func(int val) {
        // Changing the value
        val = 123;
}

int main() {
        int x = 1;

        // Passing x by value to func()
        func(x);
        printf("%d", x);
        return 0;
}
```

```
1
```



EXPLAIN

Passing **by reference** variables to functions

- ✓ Arrays are **passed by reference**.
- ✓ The address of the array is passed to the function.
- ✓ The function **works on this reference**, so it will affect the original variable.

Click Here → MORE INFO

```
void fun(int values[])
{
    values[1] = 6;
}

int main()
{
    int x[] = {40, 30, 20, 10};

    fun(x);

    printf("%d", x[1]);
    return 0;
}
```

What will this code output?

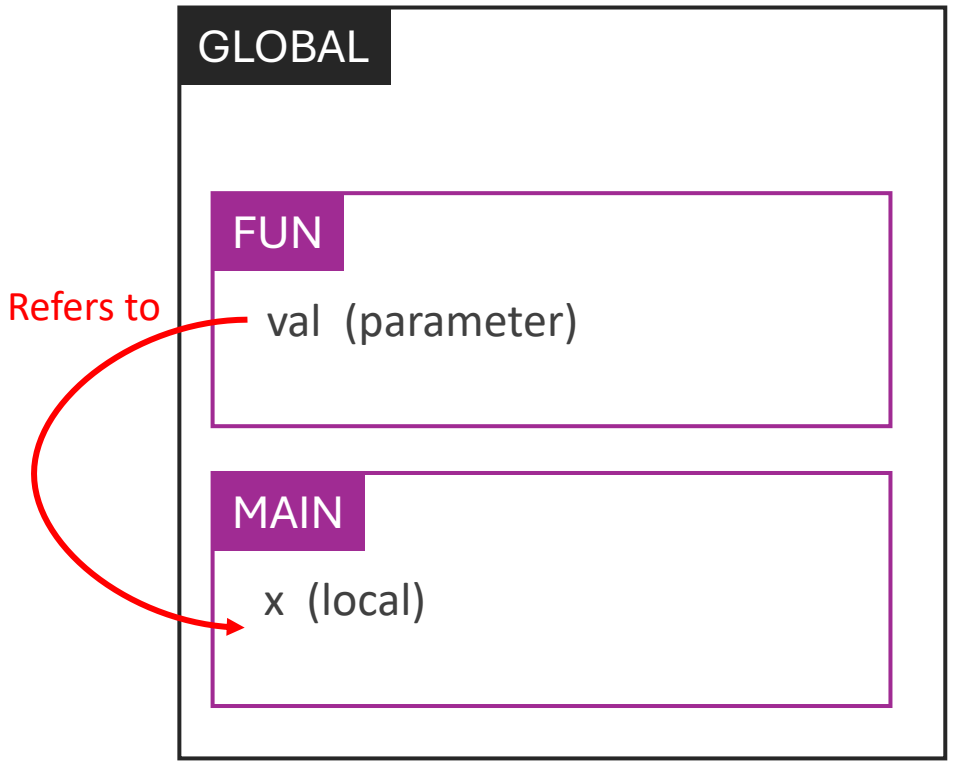
ANSWER

Passing by reference variables to functions

The variable X is affected by the function. As we pass to the function a REFERENCE of this variable

```
void fun(int values[])
{
    values[1] = 6;
}

int main()
{
    int x[] = {40, 30, 20, 10};
    fun(x);
    printf("%d", x[1]);
    return 0;
}
```

```
6
```



EXPLAIN

Passing **by value** VS **by reference**

Depending on its data type, a variable can be passed either by value or by refence

- ✓ **Primitive types** (char, bool, int, etc..) are passed **by value**
- ✓ **Complex types** (array..) are passed **by reference**

Difference	**Call-by-Value**	**Call-by-Reference**
Definition	Passes a copy of the argument value to the function.	Passes the address (reference) of the argument.
Effect on Original Argument	Original argument remains unchanged after the function call.	Original argument can be modified within the function.
Memory Usage	Requires more memory as a copy of the argument is created.	More memory-efficient as only the address is passed.

EXPLAIN

Passing **an array** to a function

✓ Whenever an array is passed to a function, it **decays into a reference** to its first element.

```
void fun(int x[])
{
    x[1] = 6;
}


int main()
{
    int x[] = {40, 30, 20, 10};

    fun(x);
    printf("%d", x[1]);
    return 0;
}
```

Sizeof(x) = **4 bytes**

The compiler only knows X is a reference
it has no idea how many elements it points to

Sizeof(x) = **16 bytes**

The compiler knows the array declaration

EXPLAIN

Passing **an array** to a function

*We need to **pass the size of the array** as second parameter to the function*

```
void fun(int x[], int size)
{
    …
}

int main()
{
    int x[] = {40, 30, 20, 10};
    fun(x, 4);

    return 0;
}
```

Now the function know the array number of elements

We pass the size of the array

QUIZ

What will this code print?

```
int x = 10;

void test() {
    int x = 5;
    if (true) {
        int x = 2;
        printf("%d ", x);
    }
    printf("%d ", x);
}

int main() {
    test();
    printf("%d ", x);
}
```

A. 2 5 10

B. 2 2 10

C. 5 5 10

D. 2 5 5

ANSWER

What will this code print?

When multiple variables with the same name exist in different scopes, the **innermost** variable is the one that will be used within that specific code block

```
int x = 10;

void test() {
    int x = 5;
    if (true) {
        int x = 2;
        printf("%d ", x);
    }
    printf("%d ", x);
}

int main() {
    test();
    printf("%d ", x);
}
```

Variable innermost!

A. 2 5 10

B. 2 2 10

C. 5 5 10

D. 2 5 5

QUIZ

Given this code, what will this expression evaluate to?

```
int a[10];

printf("%d", sizeof(a) / sizeof(int));
```

A. Size of int

B. Compiler error

C. Number of elements in a

D. Size of a in bytes

ANSWER

Given this code, what will this expression evaluate to?

```
int a[10];

printf("%d", sizeof(a) / sizeof(int));
```

A. Size of int

B. Compiler error

C. Number of elements in a

D. Size of a in bytes

QUIZ

What will this code print?

```
void modify(int a, int arr[]) {
    a = a + 10;
    arr[0] = arr[0] + 10;
}

int main() {
    int x = 5;
    int y[1] = {5};
    modify(x, y);
    printf("%d %d\n", x, y[0]);
    return 0;
}
```

A. 15 15

B. 5 15

C. 15 5

D. 5 5

ANSWER

What will this code print?

x is passed by value
=> changes inside the function do not affect the original.

arr is passed by reference,
=> modifying arr[0] does affect the original y[0]

```
void modify(int a, int arr[]) {
    a = a + 10;
    arr[0] = arr[0] + 10;
}

int main() {
    int x = 5;
    int y[1] = {5};
    modify(x, y);
    printf("%d %d\n", x, y[0]);
    return 0;
}
```

A. 15 15

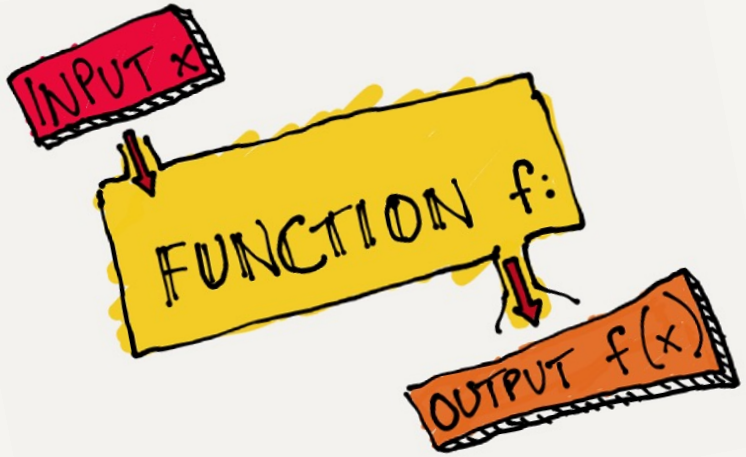
B. 5 15

C. 15 5

D. 5 5

What you **know now**

- ✓ Distinguish between **local and global** variable scopes.
- ✓ Understand variable **memory size**.
- ✓ Explain how **variables and arrays are passed to functions**.
- ✓ Be able to compute the **number of elements in array** dynamically.



Go further after the class…

Variable scope in C
https://www.w3schools.com/c/c_scope.php

Memory size in C
https://www.w3schools.com/c/c_data_types_sizeof.php

Pass by value or pass by reference
https://www.geeksforgeeks.org/parameter-passing-techniques-in-c-cpp/

Global variables in C
https://www.geeksforgeeks.org/global-variables-in-c/