User Defined Function

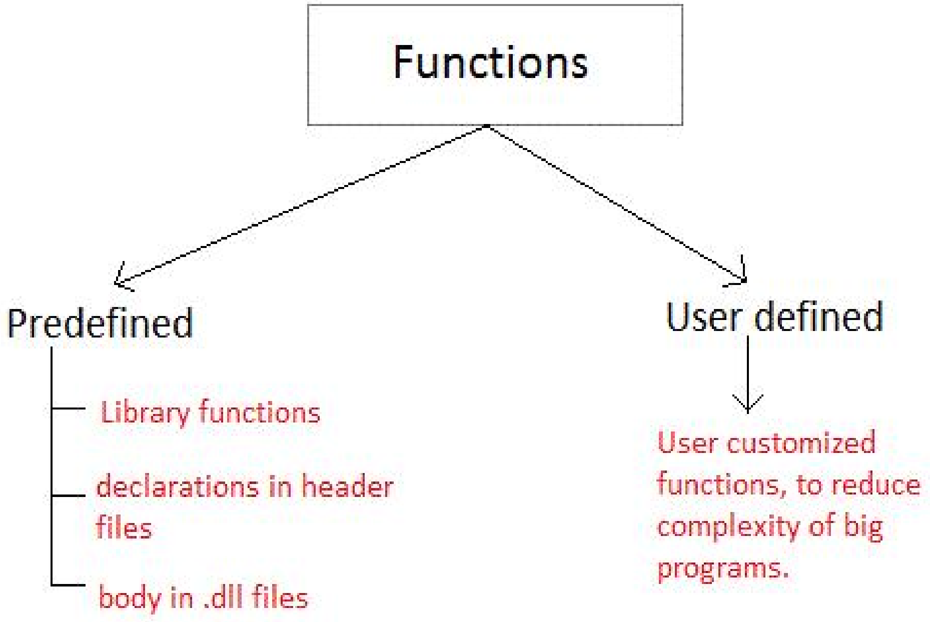
Lecture-15

Dr. Asif Uddin Khan

Functions in C

- A **function** is a block of code that performs a particular task.
- C functions can be classified into two categories,

1. **Library functions**
2. **User-defined functions**


Functions
Predefined
User defined
Library functions
declarations in header files
body in .dll files
User customized functions, to reduce complexity of big programs.

Library functions

- **Library functions** are those functions which are already defined in C library, example printf(), scanf(), strcat() etc.
- You just need to include appropriate header files to use these functions.
- These are already declared and defined in C libraries.

User-defined functions

- A **User-defined functions** on the other hand, are those functions which are defined by the user at the time of writing program.
- These functions are made for code reusability and for saving time and space.

Benefits of Using Functions

- It provides modularity to your program's structure.
- It makes your code reusable. You just have to call the function by its name to use it, wherever required.
- In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
- It makes the program more readable and easy to understand.

How to use User defined functions

Three sections of function

- Function Declaration(Function prototype)
- Function definition
- Calling a function

Function Declaration(Function prototype)

Syntax of function prototype

- returnType functionName(type1 argument1, type2 argument2, ...);

Example: int addNumbers(int a, int b);

- The above function prototype provides the following information to the compiler:

1. name of the function is addNumbers()
2. return type of the function is int
3. two arguments of type int are passed to the function
4. The function prototype is not needed if the user-defined function is defined before the main() function.

Function definition

- Function definition contains the block of code to perform a specific task.

Syntax

- returnType functionName(type1 argument1, type2 argument2, ...)
- { //body of the function
- }
- Example:

```
int addNumbers(int a, int b) // function definition
 {
int result;
result = a+b;
return result; // return statement
}
```

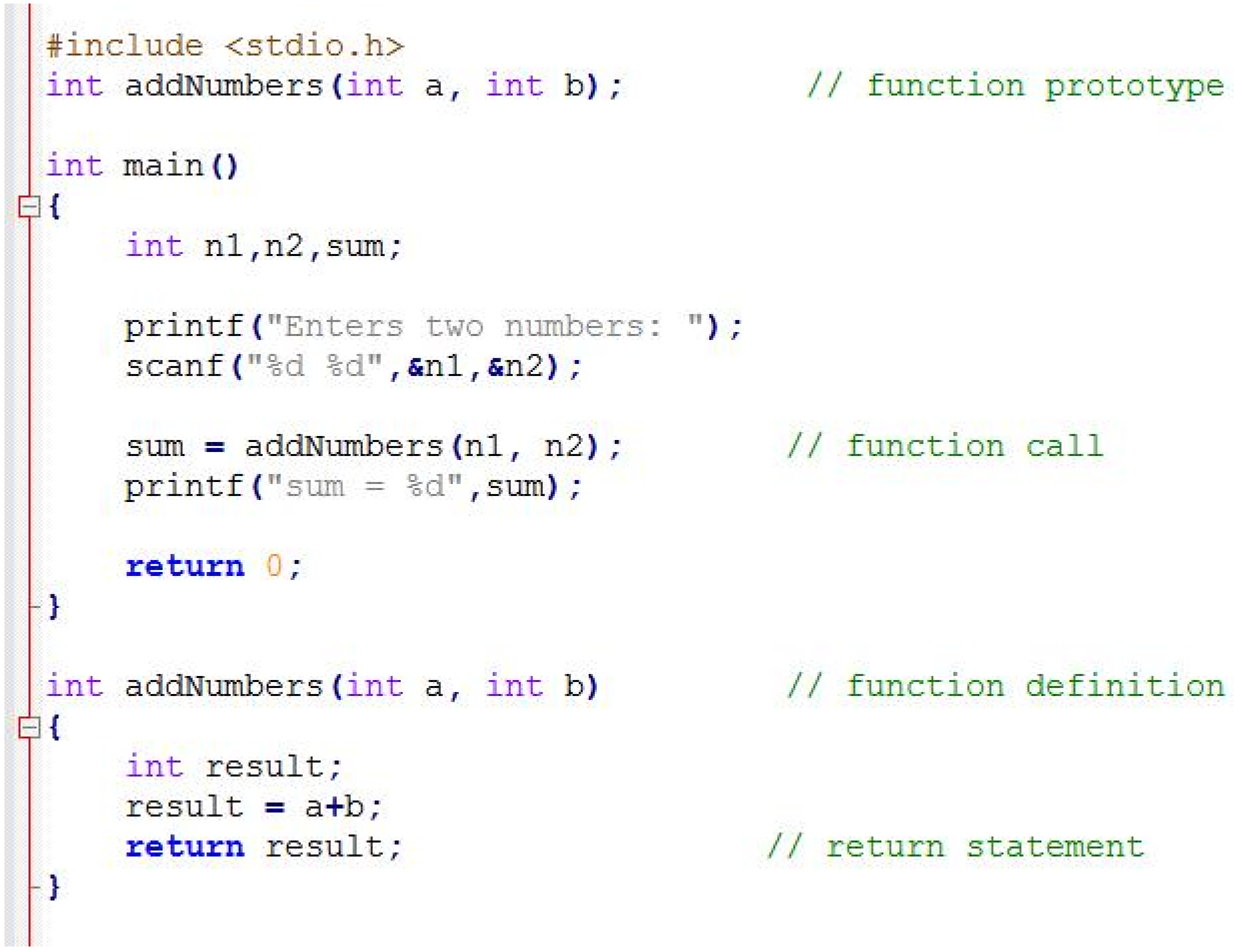
Calling a function

- Control of the program is transferred to the user-defined function by calling it.

Syntax of function call

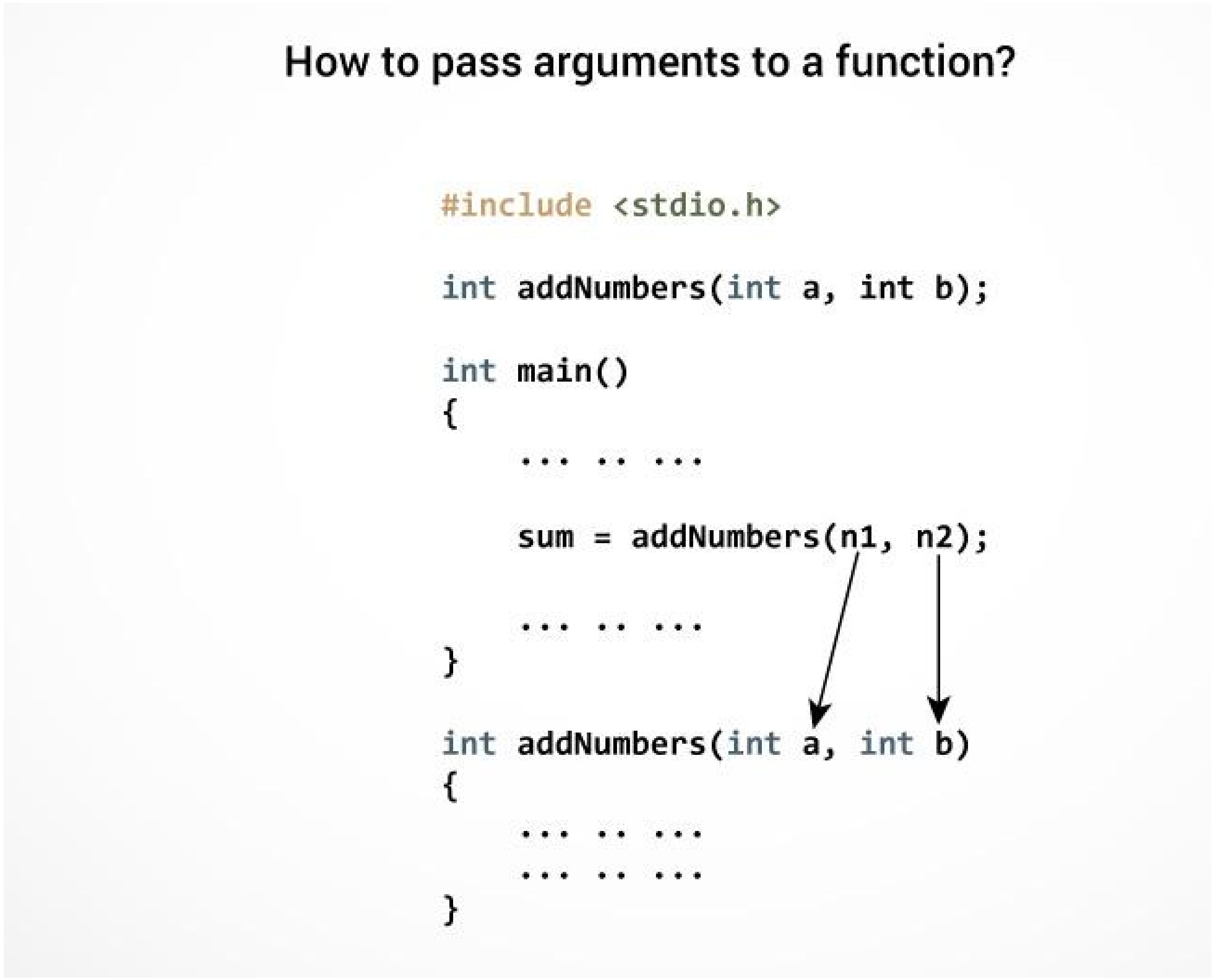
- functionName(argument1, argument2, ...);
- Example: addNumbers(n1, n2);

Example



```c
#include <stdio.h>
int addNumbers(int a, int b);         // function prototype

int main()
{
    int n1,n2,sum;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);

    sum = addNumbers(n1, n2);        // function call
    printf("sum = %d",sum);

    return 0;
}

int addNumbers(int a, int b)         // function definition
{
    int result;
    result = a+b;
    return result;                  // return statement
}
```

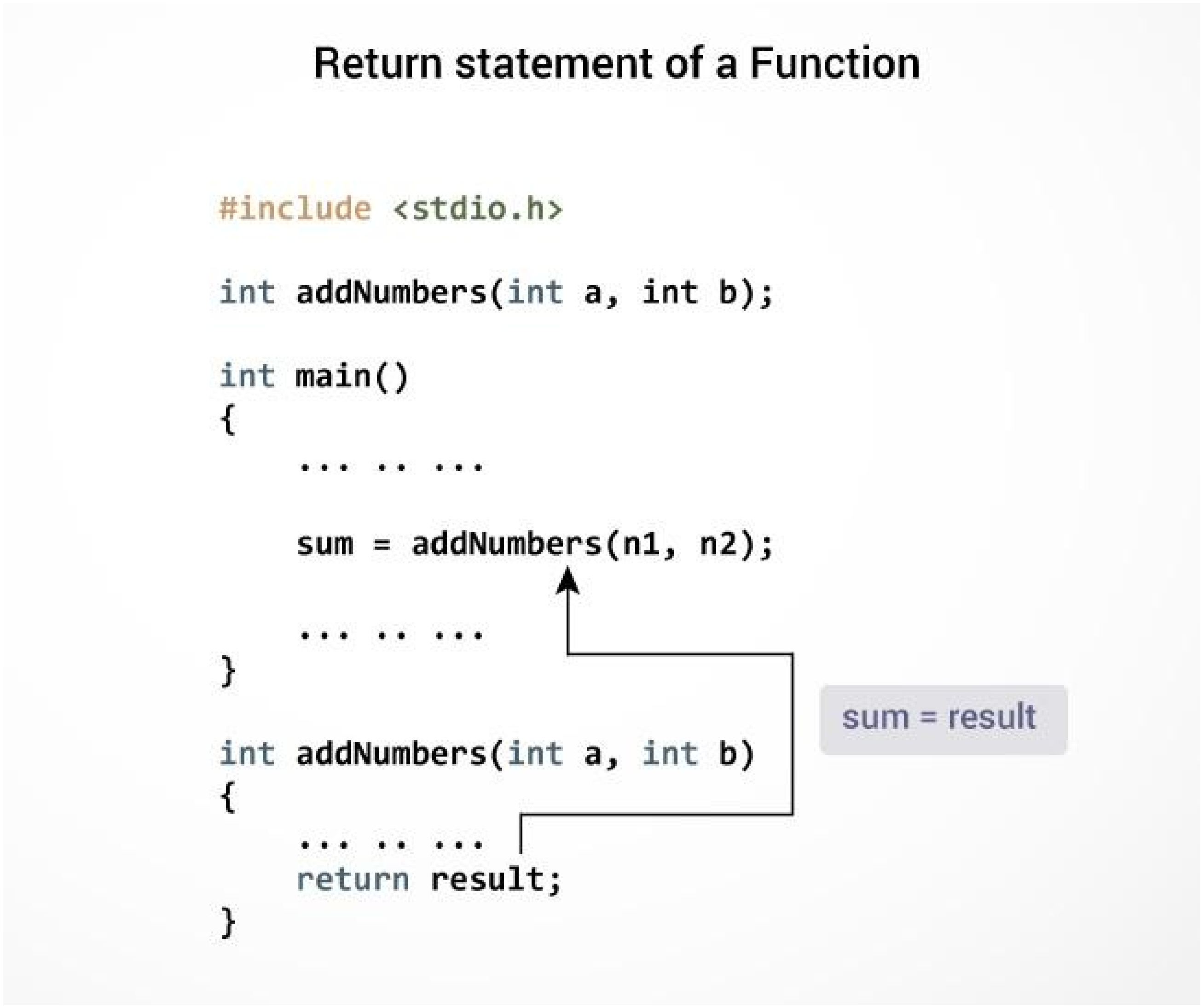
Passing arguments to a function



How to pass arguments to a function?

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    ... .. ...
}
```

Return Statement



```c
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```

Types of User-defined Functions in C

4 types

1. Function with no arguments and no return value
2. Function with no arguments and a return value
3. Function with arguments and no return value
4. Function with arguments and a return value

Function with No Arguments and No Return Value

- Used to display information or to perform any task on global variables.

```
// C program to use function with no argument and no return values
#include <stdio.h>
void sum()
{
        int x, y;
        printf("Enter x and y\n");
        scanf("%d %d", &x, &y);
        printf("Sum of %d and %d is: %d", x, y, x + y);
}
// Driver code
int main()
{
        // function call
        sum();
        return 0;
}
```

Function with No Arguments and With Return Value

- Such functions are used to perform specific operations and return their value.

```
#include <stdio.h>
int sum()
{
        int x, y, s = 0;
        printf("Enter x and y\n");
        scanf("%d %d", &x, &y);
        s = x + y;
        return s;
}
// Driver code
int main()
{
        // function call
        printf("Sum of x and y is %d", sum());
        return 0;
}
```

Function With Arguments and No Return Value

- Such functions are used to display or perform some operations on given arguments.

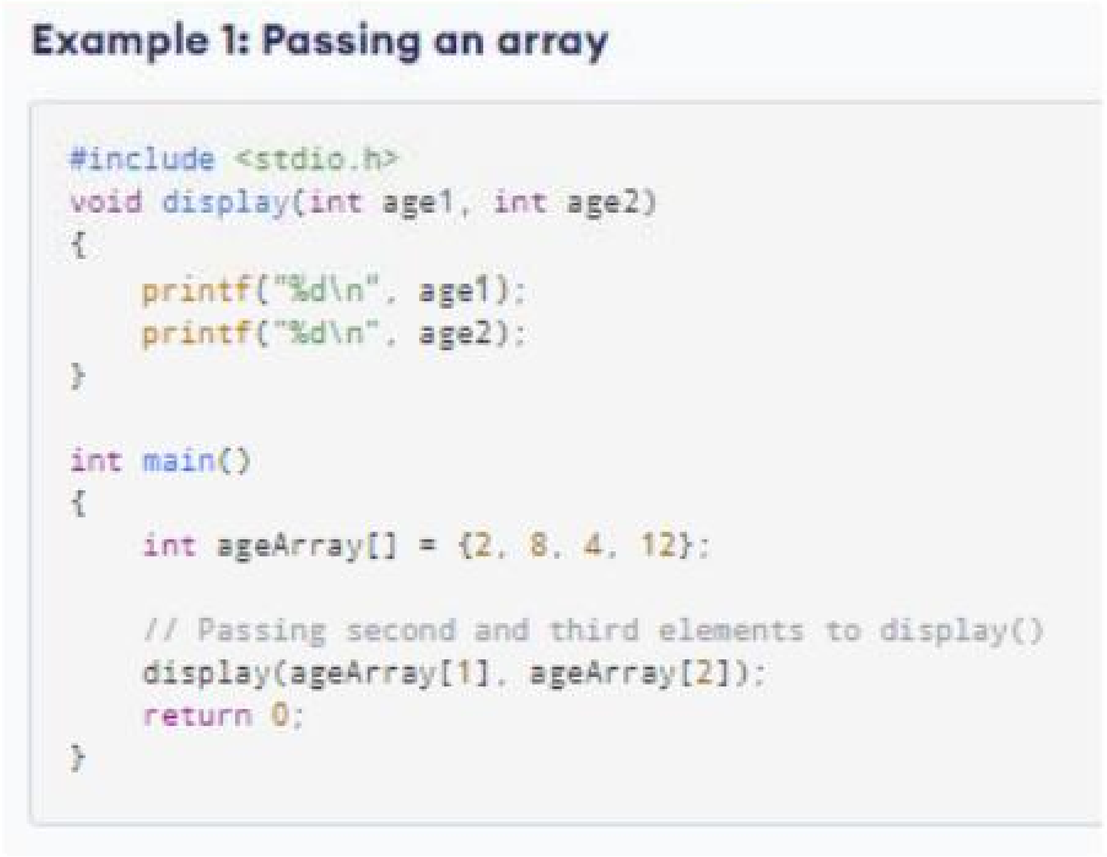
```
#include <stdio.h>
void sum(int x, int y)
{
        printf("Sum of %d and %d is: %d", x, y, x + y);
}
// Driver code
int main()
{
        int x, y;
        printf("Enter x and y\n");
        scanf("%d %d", &x, &y);
        // function call
        sum(x, y);
        return 0;
}
```

Passing arrays to a function in C

- In C programming, you can pass entire array to functions.
- Before we learn that, let's see how you can pass individual elements of an array to functions.

Passing individual array elements

- Passing array elements to a function is similar to passing variables to a function.

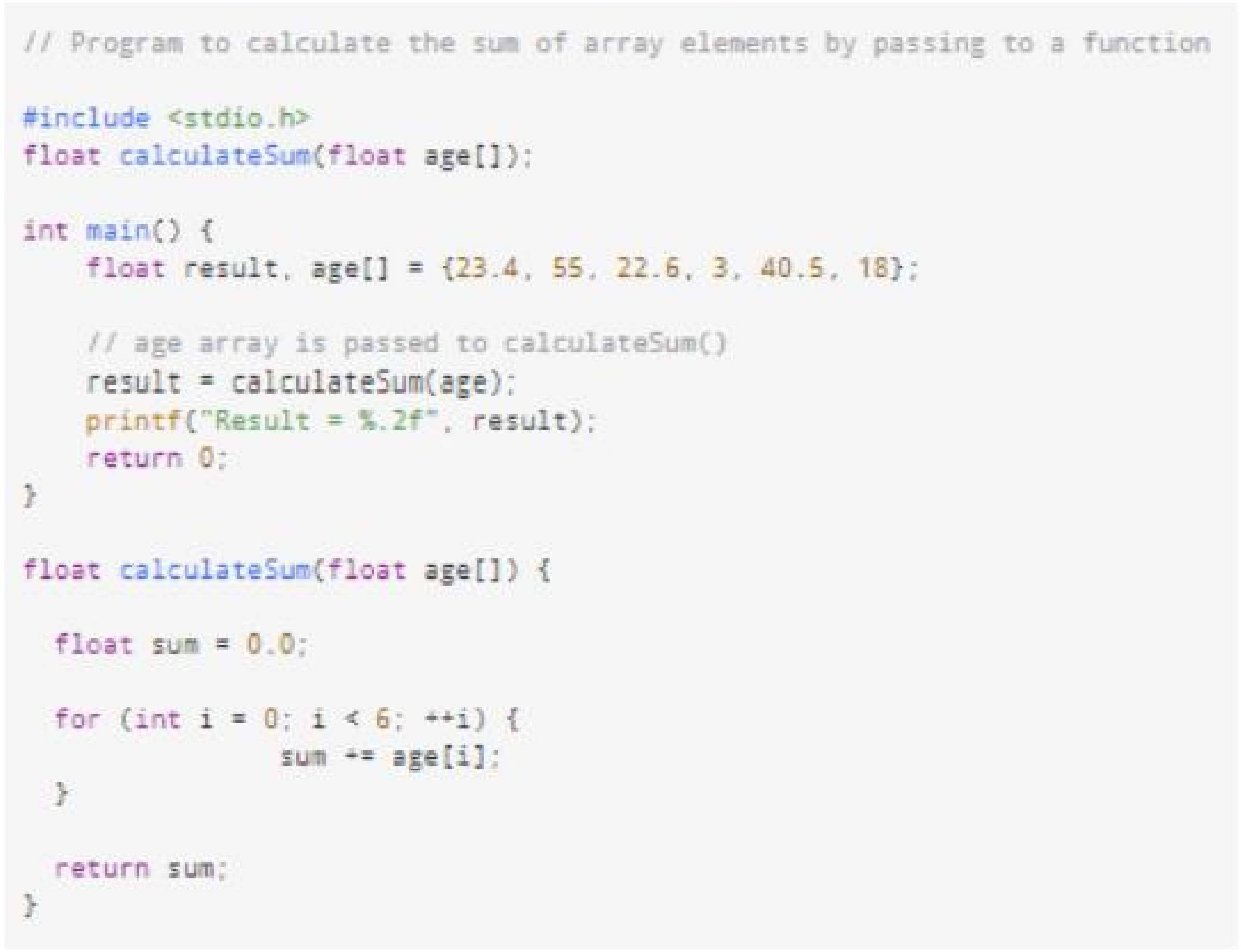


Example 1: Passing an array

```
#include <stdio.h>
void display(int age1, int age2)
{
    printf("%d\n", age1);
    printf("%d\n", age2);
}

int main()
{
    int ageArray[] = {2, 8, 4, 12};

    // Passing second and third elements to display()
    display(ageArray[1], ageArray[2]);
    return 0;
}
```

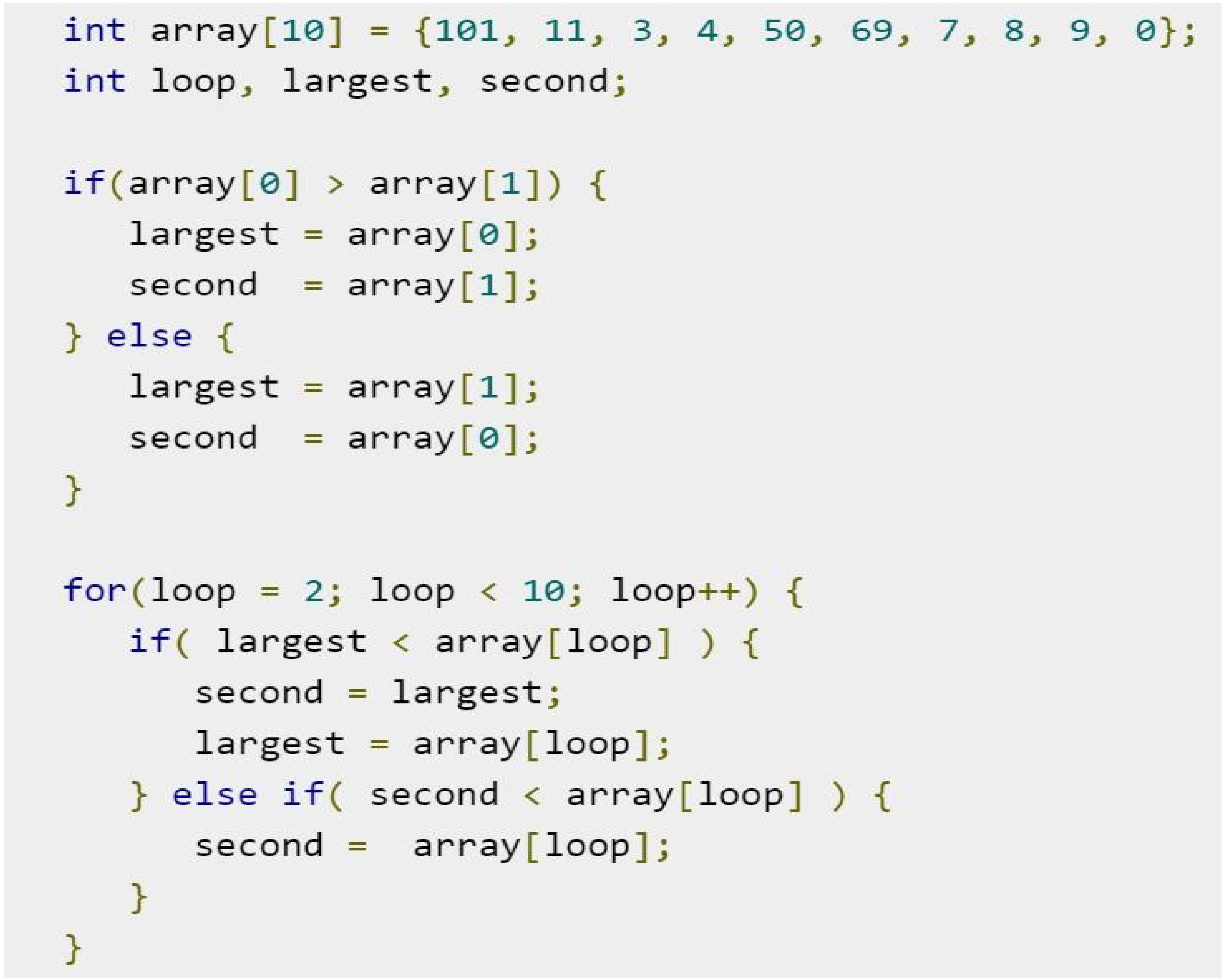
Example 2: Passing entire array to functions

- To pass an entire array to a function, only the name of the array is passed as an argument.



```c
// Program to calculate the sum of array elements by passing to a function

#include <stdio.h>
float calculateSum(float age[]);

int main() {
    float result, age[] = {23.4, 55, 22.6, 3, 40.5, 18};

    // age array is passed to calculateSum()
    result = calculateSum(age);
    printf("Result = %.2f", result);
    return 0;
}

float calculateSum(float age[]) {

  float sum = 0.0;

  for (int i = 0; i < 6; ++i) {
                sum += age[i];
  }

  return sum;
}
```

Second largest array element in C



```c
int array[10] = {101, 11, 3, 4, 50, 69, 7, 8, 9, 0};
int loop, largest, second;

if(array[0] > array[1]) {
   largest = array[0];
   second  = array[1];
} else {
   largest = array[1];
   second  = array[0];
}

for(loop = 2; loop < 10; loop++) {
   if( largest < array[loop] ) {
      second = largest;
      largest = array[loop];
   } else if( second < array[loop] ) {
      second =  array[loop];
   }
}
```

References

1. C programming by E Balaguruswami
2. Programming C by Y. kanitkar
3. Programming C by Denis Ritchie