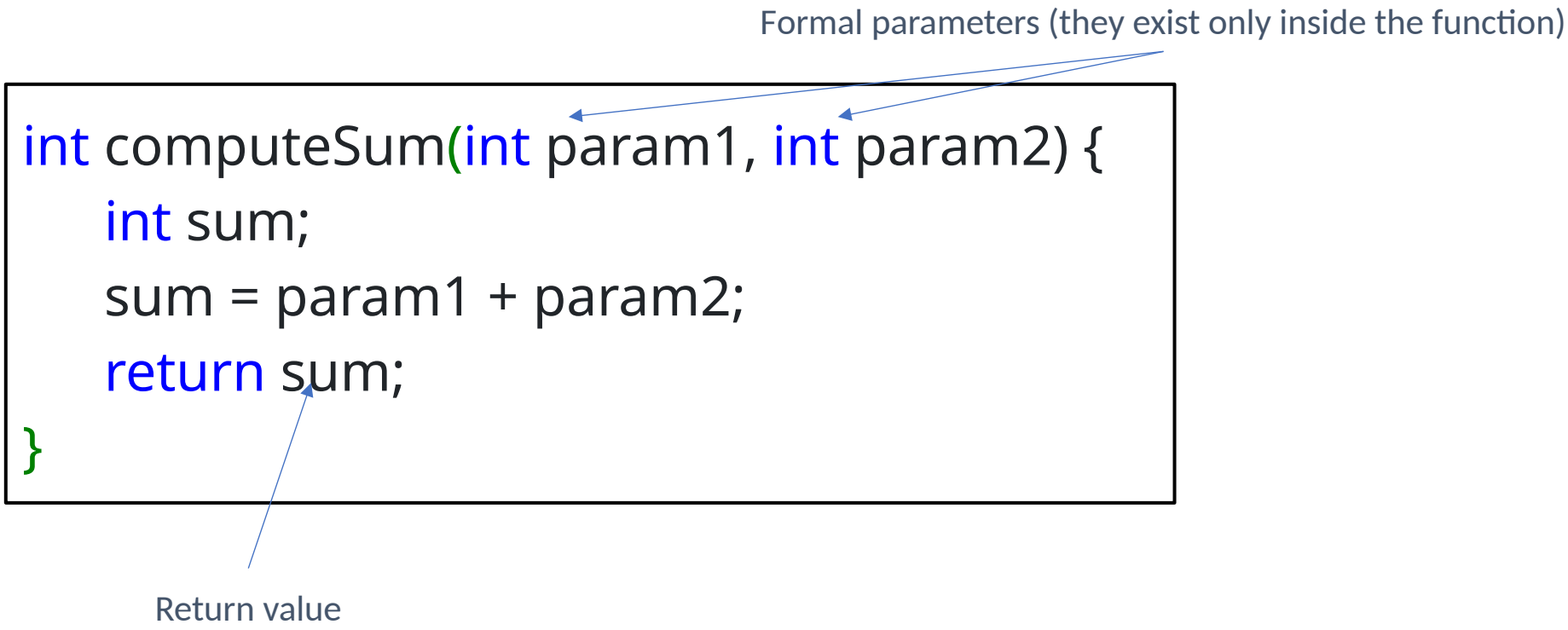
Computer Programming 1

Laboratory 06

Function

- Sub-program with a well-defined functionality that returns a value



Procedure

- Sub-program with a well-defined functionality ~~that returns a value~~

Formal parameters (they exist only inside the procedure)

```
void printSum(int param1, int param2) {
    int sum;
    sum = param1 + param2;
    cout << sum << endl;
    return;
}
```

optional

Structure

```
1. #include <iostream>
2. using namespace std;
3.
4. int computeSum(int param1, int param2);
5.
6. int main() {
7.   int a = 1, b = 2;
8.   cout << computeSum(a, b);
9. }
10.
11. int computeSum(int param1, int param2) {
12.   int sum;
13.   sum = param1 + param2;
14.   return sum;
15. }
```

Structure

```
1. #include <iostream>
2. using namespace std;
3.
4. int computeSum(int param1, int param2);
5.
6. int main() {
7.   int a = 1, b = 2;
8.   cout << computeSum(a, b);
9. }
10.
11. int computeSum(int param1, int param2) {
12.   int sum;
13.   sum = param1 + param2;
14.   return sum;
15. }
```

Declaration

Structure

```
1. #include <iostream>
2. using namespace std;
3.
4. int computeSum(int param1, int param2);
5.
6. int main() {
7.   int a = 1, b = 2;
8.   cout << computeSum(a, b);
9. }
10.
11. int computeSum(int param1, int param2) {
12.   int sum;
13.   sum = param1 + param2;
14.   return sum;
15. }
```

Declaration

Call

Structure

```
1. #include <iostream>
2. using namespace std;
3.
4. int computeSum(int param1, int param2);
5.
6. int main() {
7.   int a = 1, b = 2;
8.   cout << computeSum(a, b);
9. }
10.
11. int computeSum(int param1, int param2) {
12.   int sum;
13.   sum = param1 + param2;
14.   return sum;
15. }
```

Declaration

Call

Definition

Example 1: Print uppercase

- Given a character in input, write a program that checks - with a function named *checkCharacter* - whether the character provided in input is a lowercase letter of the alphabet. If so, the program must print the uppercase character on the screen, using another function for the conversion (named *convertCharacter*).

E.g., 'a' => 'A'

Types of implementations:
(1) Without using functions in the cctype library (e.g., isalnum, isalpha, islowwer, tolower, etc.)
(2) By using functions of the cctype library (e.g., isalnum, isalpha, islowwer, tolower, etc.)

Example 2: Print uppercase v2

- Given a character in input, write a program that checks - with a function - whether the character provided in input is a lowercase letter of the alphabet. If so, the program must print the uppercase character on the screen, using a procedure for the conversion.

E.g., 'a' => 'A'

Types of implementations:
(1) Without using functions in the cctype library (e.g., isalnum, isalpha, islowwer, tolower, etc.)
(2) By using functions of the cctype library (e.g., isalnum, isalpha, islowwer, tolower, etc.)

Example 3: Revert number

- Write a program that takes a number (or more than one digit) as input and prints the inverted number on the screen using a procedure, like in the following examples:

E.g.,

345 → 543

1234 → 4321

Example 4: Liar's Dice

- Write a program that:
  - it rolls 10 6-sided dices by using the function *rand()* as presented below.
  - then the program asks the user to guess how many dices turned out to be 1.
  - If the number specified by the user corresponds to the number of 1 of the random dices, the user is the winner, and a message is printed.
  - Otherwise, the number inserted by the user is wrong, a message is printed and the program execution ends

```
#include <cstdlib>
srand(time(NULL));          //seed value
int random_number = rand() % 6 + 1
```

For additional details:

http://www.cplusplus.com/reference/cstdlib/rand/

Example 5: RisiKo! 1vs1

- Write a program that simulates a 1-on-1 attack on RisiKo!. Roll one 6-sided dice for the attacker and one dice for the defender. Finally, declare the highest number as the winner.

Defender

Striker

The winner is: the striker!

Example 6: RisiKo! 2vs2

- Write a program that simulates a 2-on-2 attack on RisiKo!. Roll 2 6-sided dices for the attacker and 2 dices for the defender. Compare the highest die of the attacker against the highest of the defender, and the lowest of the attacker against the lowest of the defender. Finally, declare the clashes won by the attacker and those won by the defender.

```
#include<algorithm>
int maximum = max(n1, n2)
int minimum= min(n1, n2)
```

For additional information:
http://www.cplusplus.com/reference/algorithm/