6 – Exercises on Functions

Université Jean Monnet – Saint-Étienne 2026/2027

Quick reference

Listing 1:

```
// A function signature: return_type name(parameter_type parameter, ...)
int clamp(int value, int low, int high); // declaration (prototype)

// Definition
int clamp(int value, int low, int high) {
    if (value < low) return low;
    if (value > high) return high;
    return value;
}

// Overloading: same name, different parameter list
float clamp(float value, float low, float high) { ... }
void  clamp(int& value, int low, int high)  { ... }
```

Key vocabulary:

- **Signature** — the name plus the list of parameter types (the return type is *not* part of the signature).
- **Return type** — the type written before the function name; `void` means the function returns nothing.
- **Overload** — two functions with the same name but different signatures; the compiler picks the right one based on the arguments passed at the call site.
- `const` **reference** — `const int& x` gives read-only access to the caller's variable (no copy, no modification).

Part A — Starting with functions

Problem 1 — Calling functions

Jump into this Compiler Explorer session: `https://godbolt.org/z/8qfsWEW8G` to complete the exercise.

Write the main body so that the correct `print_...` function is called depending on the value of the variable `genre`. Changing the value of `genre` should change the output of the program accordingly.

Problem 2 — Writing functions

Building on the previous code, write a function `void print_genre(Genre genre)` that prints the music genre of the variable passed as argument. Use that function to simplify your previous main function.

Part B — Arguments and references

Problem 1 — Pass by value vs. pass by reference

What does the program in Listing 2 print?

Listing 2: Value vs. reference

```
#include <iostream>

void boost_v1(int volume, int amount) {
    volume = volume + amount;
}

void boost_v2(int& volume, int amount) {
    volume = volume + amount;
}

int main() {
    int vol1 = 50;
    int vol2 = 50;

    boost_v1(vol1, 20);
    boost_v2 (vol2, 20);

    std::cout << vol1 << std::endl; // line A
    std::cout << vol2 << std::endl; // line B
    return 0;
}
```

Line A:

Line B:

Why are the two results different?

Problem 2 — `const` reference

Does the program in LISTING 3 compile? If not, identify the line that causes the error and explain why.

Does it compile?

If not, why?

Listing 3: Const reference

```
1  #include <iostream>
2
3  void print_brightness(const int& value) {
4      std::cout << value << std::endl;
5      value = 100;
6  }
7  int main() {
8      int b = 200;
9      print_brightness(b);
10     return 0;
11 }
```

Problem 3 — Write your own

Write a function that computes the average of two floating numbers and stores the result in a variable given as the first argument. The caller should be able to give optional weights for each value. The function should respect the following:

- The weights have to be in range [0, 1].
- The sum of both weight should be 1.
- If either one of the previous two rules is not respected, the function should store -1 in the result variable.
- If the weights are not provided by the user, they should each be equal to `0.5`.

Some calling example:

Listing 4:

```cpp
int main() {
    float result;
    average(result, 12, 16);
    std::cout << result << std::endl;

    average(result, 12, 18, 0.3);
    std::cout << result << std::endl;

    average(result, 12, 16, 2);
    std::cout << result << std::endl;

    average(result, 12, 16, 0.3, 0.6);
    std::cout << result << std::endl;

    average(result, 12, 16, 0.2, 0.8);
    std::cout << result << std::endl;
}
```

Expected output:

Listing 5:

```
14
16.2
-1
-1
15.2
```

Note: As a reminder, the weighted average for two numbers a and b of respective weights w_a and w_b is:

$$\frac{a \times w_a + b \times w_b}{w_a + w_b}$$

Part C — Reading signatures

Problem 1 — Identify the parts

For each function below, fill in the table.

Listing 6:

```cpp
bool  is_muted(int volume);
void  set_volume(int& volume, int target);
int   mix(int track_a, int track_b);
float pan(float signal, float amount);
void  print_genre(const std::string& name);
int   clamp_brightness(int value, int lo, int hi);
```

Function	Return type	Parameter types	Returns a value?
`is_muted`			
`set_volume`			
`mix`			
`pan`			
`print_genre`			
`clamp_brightness`			

Problem 2 — Match the call to the signature

The four overloads below all exist at the same time.

Listing 7: Four overloads of `describe`

```
// (A)
void describe(int level);
// (B)
void describe(int level, bool verbose);
// (C)
void describe(float level);
// (D)
void describe(const int& level);
```

In Table 1, for each call of the first column, write the letter of the overload selected by the compiler, and the return type of that call.

Table 1: Answers

Call	Overload chosen	Return type
`describe(80, false)`		
`describe(80)`		
`describe(0.5f)`		
`describe(80, true)`		

Note: an undecorated integer literal such as `80` is of type `int`. The compiler prefers an exact match over a `const` reference when both are available.

Part D — Return values

Problem 1 — Trace the return value

What does `main` print in LISTING 8?

Line A:

Line B:

Line C:

Line D:

Listing 8: Note classifier

```cpp
#include <iostream>

int note_value(int note_number) {
    if (note_number < 1 || note_number > 7)
        return -1;
    return note_number * 10;
}

bool is_valid(int note_number) {
    return note_number >= 1 && note_number <= 7;
}

int main() {
    std::cout << note_value(3) << std::endl; // line A
    std::cout << note_value(9) << std::endl; // line B
    std::cout << is_valid(5) << std::endl;  // line C
    std::cout << is_valid(0) << std::endl;  // line D
    return 0;
}
```

Problem 2 — What is the return type?

For each function body below, write the correct return type (choose among `int`, `bool`, `float`, `void`).

Listing 9: Missing return types

```
??? brightness_average(int r, int g, int b) {
    return (r + g + b) / 3;
}

??? is_greyscale(int r, int g, int b) {
    return r == g && g == b;
}

??? print_pixel(int r, int g, int b) {
    std::cout << r << " " << g << " " << b << std::endl;
}

??? pan_value(int signal, float amount) {
    return signal * amount;
}
```

`brightness_average`

`is_greyscale`

`print_pixel`

`pan_value`

For `pan_value`: what type does `int * float` produce in C++?

Part E — Overloading

Problem 1 — Overload by argument type

Here is a piece of code:

Listing 10: Overloaded `invert`

```
#include <iostream>

int   invert(int value) { return 255 - value; }
float invert(float value) { return 1.0f - value; }
bool  invert(bool value) { return !value; }

int main() {
    std::cout << invert(200) << std::endl; // line A
    std::cout << invert(0.3f) << std::endl; // line B
    std::cout << invert(true) << std::endl; // line C
    std::cout << invert(invert(60)) << std::endl; // line D
    return 0;
}
```

For each line A, B, C and D, complete TABLE 2 with the chosen overload and the value returned by `invert`.

Table 2: Answers

Line	Overload chosen	Returned Value
Line A		
Line B		
Line C		
Line D		

Problem 2 — Overload by argument count

Here is a piece of code:

Listing 11: Overloaded `mix`

```cpp
#include <iostream>

int mix(int a, int b) {
    return (a + b) / 2;
}

int mix(int a, int b, int c) {
    return (a + b + c) / 3;
}

int mix(int a, int b, int c, int d) {
    return (a + b + c + d) / 4;
}

int main() {
    std::cout << mix(40, 60)       << std::endl; // line A
    std::cout << mix(10, 20, 30)   << std::endl; // line B
    std::cout << mix(0, 100, 50, 150) << std::endl; // line C
    std::cout << mix(mix(0, 100), mix(50, 150)) << std::endl; // line D
    return 0;
}
```

Complete TABLE 2 with the value printed on each line A, B, C and D.

Table 3: Answers

Line	Printed Value
Line A	
Line B	
Line C	
Line D	

Problem 3 — Overload by `const` reference

Listing 12: Overload on constness

```
#include <iostream>

void inspect(int& value) {
    std::cout << "non-const: " << value << std::endl;
    value = value + 1;
}

void inspect(const int& value) {
    std::cout << "const: " << value << std::endl;
}

int main() {
    int       a = 80;
    const int b = 120;

    inspect(a);  // line A
    inspect(b);  // line B
    inspect(50); // line C

    std::cout << a << std::endl; // line D
    return 0;
}
```

Why can't line C call the non-`const` overload?

Table 4: Answers

Line	Used Overload	Printed Value
Line A		
Line B		
Line C		
Line D	(no call to `inspect`)	

Part F — Spot the bug

Problem 1 — Three buggy functions

Each function below has one bug. Identify each bug and write a fix. You can use the Godbolt links below to write your fix, and then copy-paste the shared link (Go to Share in the upper right corner).

Listing 13: Bug hunt

```
// (1) Should return true when volume is between 0 and 100 inclusive.
bool is_valid_volume(int volume) {
    if (volume >= 0 && volume <= 100)
        return true;
}

// (2) Should double the brightness of a pixel in place.
void double_brightness(const int& value) {
    value = value * 2;
}

// (3) Should compute the average of two ints as a float.
float average(int a, int b) {
    return a + b / 2;
}
```

Bug 1:

Fix 1: `https://godbolt.org/z/ccTYraEPM`

Bug 2:

Fix 2: `https://godbolt.org/z/xqqcdo856`

Bug 3:

Fix 3: `https://godbolt.org/z/ff7exsc6r`

Note: to convert a numeric value to a certain type, you can use `static_cast<target_type>(my_var)`.

Part G — Write the code

Problem 1 — Image utilities (mixing all notions)

Write a program containing the following four functions and a `main` function.

Function 1 — `clamp`
Signature: `int clamp(const int& value, const int lo, const int hi)`
Returns `lo` if `value < lo`, `hi` if `value > hi`, otherwise returns `value`.

Function 2 — `clamp` **(overload)**
Signature: `void clamp(int& value, const int lo, const int hi)`
Same logic as above, but modifies `value` in place and returns nothing.

Function 3 — `is_valid_pixel`
Signature: `bool is_valid_pixel(int r, int g, int b)`
Returns `true` only if all three values are between 0 and 255 inclusive.

Function 4 — `print_pixel`
Signature: `void print_pixel(const int& r, const int& g, const int& b)`
Prints the three values on one line separated by spaces.

In `main`:

1. Call `clamp(300, 0, 255)` (value overload) and print the result.
2. Declare `int r = 300, g = 150, b = -10`
   (a) Call the reference overload of `clamp` on each with `lo=0, hi=255`;
   (b) Call `print_pixel(r, g, b)`.
3. Print the result of `is_valid_pixel(r, g, b)`.

Expected output:

Listing 14:

```
255
255 150 0
1
```

Link to your solution :

Problem 2 — Music utilities

Write a program containing the following functions and a `main` that calls each of them.

Function 1 — `tempo_label`

Signature: `void tempo_label(int bpm)`

Prints a label based on the value of `bpm`:

Table 5: BPM Labels

Condition	Label
`bpm < 60`	`"slow"`
`60 <= bpm < 120`	`"medium"`
`120 <= bpm < 180`	`"fast"`
otherwise	`"very fast"`

Function 2 — `tempo_label` **(overload)**

Signature: `void tempo_label(int bpm, bool verbose)`

When `verbose` is `true`, prints `"bpm: N -- label"`, where N is the bpm value and label is the label described in Table 5. When `verbose` is `false`, behaves exactly like the one-argument version.

Hint: you can call the one-argument overload from inside this one.

Function 3 — `beats_in_bars`

Signature: `int beats_in_bars(int bars, int beats_per_bar)`

Returns the total number of beats in `bars` bars.

Function 4 — `scale_bpm`

Signature: `void scale_bpm(int& bpm, float factor)`

Multiplies `bpm` by `factor` in place, storing the result as an `int` (truncation is acceptable).

In `main`:

1. Declare `int bpm = 90`.
2. Call `tempo_label(bpm)`.
3. Call `tempo_label(bpm, true)`.
4. Print `beats_in_bars(4, 4)`.
5. Call `scale_bpm(bpm, 1.5f)` then print `bpm`.
6. Call `tempo_label(bpm, true)`.

Expected output:

Listing 15:

```
medium
bpm: 90 -- medium
16
135
bpm: 135 -- fast
```

Link to your solution :
