automatic local variable: A variable that is created when a function is called and destroyed when the function terminates.

default parameter: A parameter value that is used when an argument is omitted in a function call.

function: A named unit of computation. It executes a block of code.

function body: The block that defines the action to be performed by a function.

function declaration: In C++, a prototype statement that describes a function's return type and formal parameters.

function definition: In C++, the implementation of a function declaration.

function header: In a function definition, that part of the function that supplies the return type, function identifier, and formal parameters.

function overloading: The definition of two or more functions within the same scope using the same identifier with different argument lists.

function prototype: function declaration.

function scope: The range from the point at which a function is declared to the end of the program.

function shadowing: The visibility of the entity in an inner block overrides the visibility of the entity in the outer block.

function signature: The parts of the function declaration that the compiler uses to perform overload resolution.

global scope: A scope that is outside any local scope.

lifetime: The processing period in which a variable can be referenced within a function.

local scope: Synonym for a block scope.

nested block: A block within another block.

parameter list: A possible empty list of parameters in the definition of a function.

pass-by-pointer: Passing the address of an object to a called function.

pass-by-reference: How arguments are passed to a function using references.

pass-by-value: How arguments are passed to a function by copying values.

scope: A part of a program in which names have meaning.

static local variable: A variable that exists for the duration of the program.

void function: Functions whose return value is void.

A **function** is an entity with a header and a set of statements that are designed to do a task.

As programs become more complex, we divide the task into smaller tasks, each of which is responsible for a part of the job.

There are four benefits to dividing a task into several small tasks: easier to write, easier to debug. reusable, and can be saved for future use.

Three entities are associated with a function: **function definition**, **function declaration**, and **function call**.

In a library function, the definition is already created in the C++ library; we only need to include the corresponding header file in our program and then call the function.

In a user-defined function, we declare, define, and call the function.

Functions in a program may need to exchange data. We refer to this activity as data pass and data return. Each can be done using one of the three methods: **by value**, **by reference**, or **by pointer**.

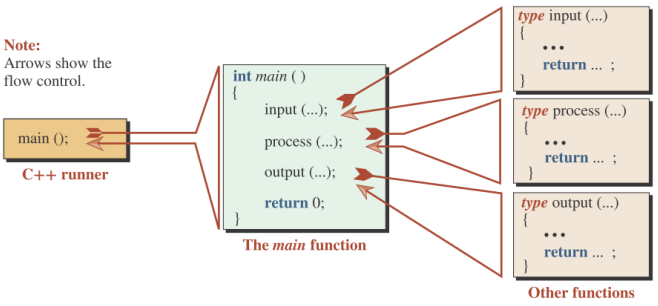
We can have one or more default parameters with predefined values in a function declaration.

Function overloading provides several definitions for a function with the same name but with different signatures, which means a different number of parameters, or different types of parameters, or both.

The return value of a function is not part of its signature.

Scope and **lifetime** are two concepts that affect the design and use of functions. Scope defines where in the source code a named entity is visible. An entity in a program has a lifetime: It is born and it dies.

CP-1. The *main* function and any other than functions in a program are called by what entity?

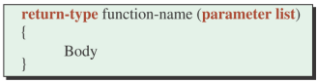


CP-2. List some benefits of creating functions in a program and dividing the task between them instead of writing the whole task to be done in the main function.

- Easier to Write Simpler Task
- Error Checking (Debugging)
- Reusability
- Library of Functions

CP-3. What is the difference between a function definition, a function declaration and a function call?

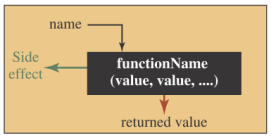
The **function definition** creates the function and is made of two sections: the *function header* (data type, name of function, parameter list) and the *function body* (algorithm).



The **function declaration** (function prototype) is the header of the function followed by a semicolon. Types are required and parameters names are optional. Is used to show how the function should be called.

```
int larger (int first, int second);          // With the names of the parameters
int larger (int, int );                      // Without the names of the parameters
```

The **function call** is a postfix expression that invokes a function to do its job. It can have a side effect, a return value, or both



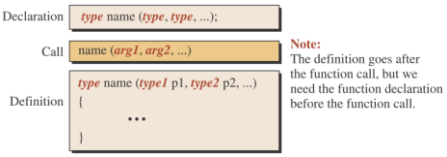
Note:
A function call can have a side effect, a return value, or both.

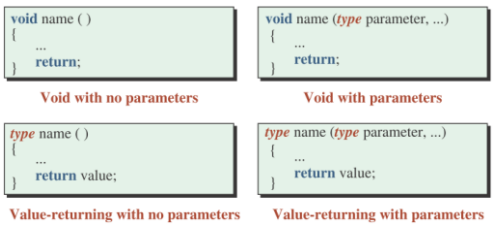
CP-4. What is the difference between a library function and a user-defined function?

The **library function** is a predefined function in library, where we do not need to create a definition. We only add the corresponding header file (declaration) in which the function is defined.

```
#include <library name>
```

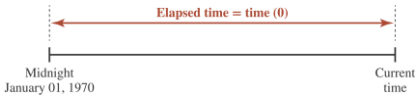
The **user-defined function** is a new function not defined in the C++ library. To call the function we must first define it in our program. Before calling a function we need to declare it or alternatively write a definition before declaration rather than after.





CP-5. What is the unit number returned from the *time(0)* function?

It gives the count of seconds from the epoch until the time that the function is called. This is Greenwich mean time (GMT), not the local time.

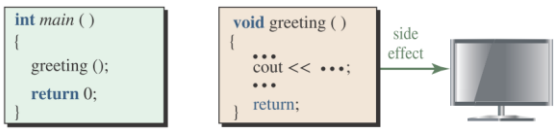


CP-6. Show how to create a random number between 10 and 20.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
int main()
{
        int low = 10;
        int high = 20;
        srand(time(0));
        int temp = rand();
        int randomNumber = temp % (high – low + 1) + low;
        std::cout << "Random number between 10 and 20: ";
        std::cout << randomNumber;
        return 0;
}
```

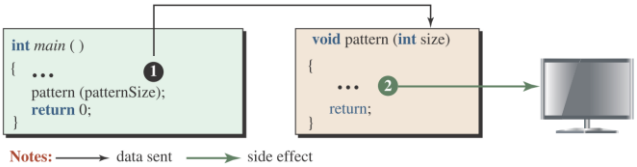
CP-7. Give the purpose of a void function with no parameters

Is only defined for its side effect, which occurs inside the function; otherwise, the function is useless.



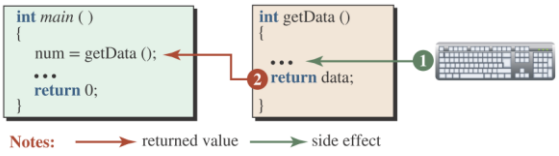
CP-8. Give the purpose of a void function with parameters.

The function has a side effect but does not return data to its caller. This type of function is used when we need only the side effect of the function but we also need to pass some information, such as what is to be output each time we call the function.



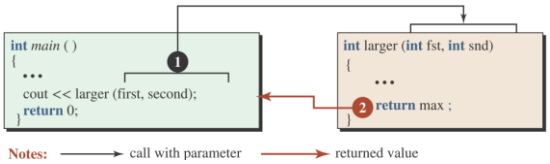
CP-9. Give the purpose of a value-returning function with no parameters.

It is designed only for its return value. The function normally has an input side effect and returns the input value to the calling function.



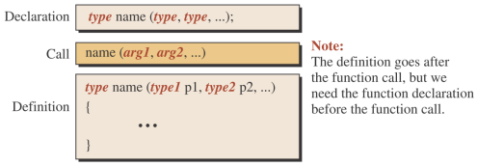
CP-10. Give the purpose of a value-returning function with parameters.

We pass some information inside the function to get the return value.



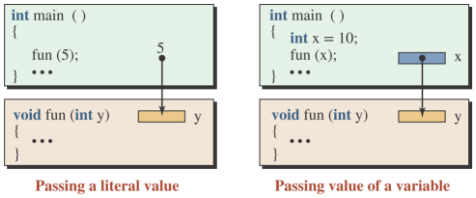
CP-11. What is the application of a function declaration in a program?

Function must be defined before the function call. Another option is to declare the function (semicolon at the end) before calling and define it after calling. This solution shorten the main code significantly.



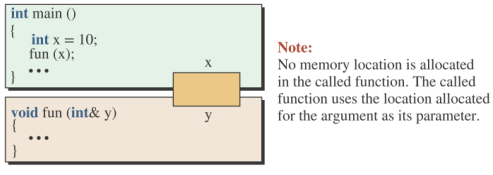
CP-12. Can the called function change the value of the corresponding argument in the calling function when data exchange is pass-by-value?

No. The pass-by-value method initializes a parameter with a copy of the corresponding argument. Called function can only read the value of the arguments. We are using it if we don not want to change the value of passed arguments.



CP-13. Can the called function change the value of the corresponding argument in the calling function when data exchange is pass-by-reference?

Yes, since a memory location is shared between the argument and the corresponding parameter. The called function can rad the value of the arguments in the calling function, and it can also change them.



CP-14. Give the advantages and disadvantages of pass-by-value data exchange.

Advantages:

- simple
- protects arguments from being changed by the called function

Disadvantages:

- the copy consumes more memory (especially large object), that is why this method is not used in OOP

CP-15. Give the advantages and disadvantages of pass-by-reference data exchange.

Advantages:

- the best choice for swapping the value
- copy of the file is not needed (low memory consumption)

CP-16. When do we need to use return-by-reference?

When we want to return large object in OOP. This solution is more efficient because no copy is created and the object does not exist anymore when the function terminates.

CP-17. Can we have a default parameter when using pass-by-reference?

No. Default parameter can only be used in pass-by-value, because default parameter cannot change the value of argument.

CP-18. What is the difference between an automatic local variable and a static local variable? Where should we use each?

An **automatic local variable** is born when the function is called and dies when the function terminates. By default all local variables in a function have automatic lifetime, but we can explicitly use the modifier *auto* in front of the variable declaration to emphasize that local variables are reborn each time the function is called.

A **static local variable** is created by using the static modifier. The lifetime of a static variable is the lifetime of the program in which it is defined. A static variable is initialized only once, but the system keeps track of its contents as long as the program is running. This means it is initialized in the first call to the function, but it can be changed during subsequent function calls.