5.1 Basic syntax and structure of a C++

The basic syntax of C++ consists of the following five elements:

• Variables: Variables are used to store data, such as numbers and strings. They are declared with a type, such as int or char, followed by an identifier, and must be initialized with a value.

• Operators: Operators are used to perform operations on variables, such as addition, subtraction, multiplication, and division.

• Expressions: Expressions are used to combine operators and variables and are evaluated by the compiler to produce a single value.

• Control Structures: Control structures are used to control the flow of the program and include if-else statements, loops, and switch statements.

• Functions: Functions are reusable blocks of code that can be used to perform tasks. A function must be declared before it is used.

Here is an example of a basic C++ program that blinks LED on a 13-th pin of an Arduino Uno controller:

Program 1: Native C++ program for ATmega328.

```
#include <avr/io.h>
#include <util/delay.h>

// Function declaration
void setup();

int main()
{
    // Variable declaration
    int time_ms = 1000;
    // Function call
    setup();
    // Main LOOP program instructions
    while (true)
    {
        PORTB |= (1<<PINB5);
        _delay_ms(time_ms);
        PORTB &= !(1<<PINB5);
        _delay_ms(time_ms);
    }
    return 0;
}
// Function definition
void setup() {
    PDDB |= (1<<PINB5);
}
```

Programming an Arduino Uno board in native C++ is much more difficult than in Arduino IDE. Arduino IDE makes it easier for users to write and debug code without having to know the details of the underlying hardware. In addition, the IDE provides many additional functions which simplify the usage of additional peripherals and actuators such as serial communication, LCDs, servo motors, step motors… This is especially true and important for beginners.

5.1.1 Tasks:

1. <++>
2. <++>
3. <++>

5.1.2 Questions:

1. <++>
2. <++>
3. <++>

5.1.3 Summary:

5.1.3.1 <++>

5.1.4 Issues:

5.1.4.1 Not including a semicolon at the end of each statement: Every statement in C++ must end with a semicolon. If a semicolon is omitted, the code will not compile correctly.

5.1.4.2 Not properly formatting the code: Properly indenting and spacing code is important in C++ to make the code easier to read. Not formatting the code correctly can lead to syntactical errors.

5.1.4.3 Not using correct capitalization: C++ is a case sensitive language and therefore proper capitalization is important. If the wrong capitalization is used, it can lead to syntax errors.