**Hanami Syntaxes:**

Use the following **Keyword Conversion** table:

| **Hanami** | **C++ Equivalent** | **Meaning** |
| --- | --- | --- |
| garden <Name> | namespace <Name> { ... } | Declares a namespace (“garden”). |
| species <Name> | class <Name> | Declares a class (“species,” i.e., a “blossom”). |
| open: | public: | Public section. |
| hidden: | private: | Private section. |
| guarded: | protected: | Protected section. |
| grow <functionName>() -> <type> | <type> <functionName>() | Declares a function (“grow”). |
| blossom <value>; | return <value>; | Returns a value. |
| style <library> | #include <library> | Includes a library. |
| bloom << x; | std::cout << x; | Prints to the console. (“bloom” = console output) |
| water >> x; | std::cin >> x; | Reads from input. (“water” = console input) |
| branch (<cond>) { ... } | if (<cond>) { ... } | If‐statement. |
| else branch (<cond>) { ... } | else if (<cond>) { ... } | Else‐if. |
| else { ... } | else { ... } | Else. |

**Note**: We keep standard C++ types (int, string, bool, etc.) exactly as in C++.

**1.1 Example Code**

style <iostream>

style <string>

garden SimpleGarden

species Rose {

open:

grow sayHello() -> void {

bloom << "Hello from Hanami Rose!\n";

blossom;

}

hidden:

int secretNumber = 42;

guarded:

bool isFriendly = true;

};

grow mainGarden() -> int {

std::string userName;

bloom << "What's your name? ";

water >> userName;

Rose g;

g.sayHello();

branch (userName == "Rose") {

bloom << "You have a lovely name!\n";

}

else branch (userName == "Lily") {

bloom << "Another beautiful flower name!\n";

}

else {

bloom << "Nice to meet you, " << userName << "!\n";

}

blossom 0;

}

A hypothetical Hanami compiler would transpile this into regular C++ (replacing the custom keywords with standard namespace, class, #include <iostream>, etc.) and then compile with a normal C++ toolchain.