Name Space

Namespaces provide a method for preventing name conflicts in large projects.

Symbols declared inside a namespace block are placed in a named scope that prevents them from being mistaken for identically-named symbols in other scopes.

★Syntax

inline(optional) **namespace** *attr*(optional) *identifier* **{** *namespace-body* **}**

{Since c++17} ← optional sequence of any number of attributes

→ {Since c++17}

⇒ Namespace-body: possibly empty sequence of declarations of any kind (including class and function definitions as well as nested namespaces)

---

① **namespace** *ns_name* **{** *declarations* **}**

⇒ Named namespace definition for the namespace ns_name.

② **namespace** **{** *declarations* **}**

⇒ They are directly usable in the same program and are used for declaring unique identifiers.

⇒ The name of the namespace is uniquely generated by the compiler.

⇒ The unnamed namespaces you have created will only be accessible within the file you created it in.

⇒ Unnamed namespaces are the replacement for the static declaration of variables.

{Example}

```
// unnamed namespace declaration
namespace
{
   int rel = 300;
}

int main()
{
   cout << rel << "\n"; // prints 300
   return 0;
}
```

③ using-directive

→ any name after a using-directive and until the end of the scope in which it appears, every name from ns_name is visible

using namespace *ns_name*;

④ using-declaration

→ makes the symbol name from the namespace ns_name accessible for unqualified lookup as if declared in the same class scope, block scope, or namespace as where this using-declaration appears.

using *ns_name*::*name*;

⑤ Type alias

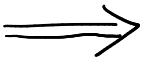
using *identifier attr*(optional) = *type-id* ;

```cpp
#include<iostream>

namespace A{
  int a = 5;
  namespace B{
    class X{
    public:
      void pp(){
        std::cout<<"I am in pp"<<std::endl;
      }
    };
  }
}

using C = A::B::X;

int main(){
  C c1;
  c1.pp();
  return 0;
}
```



```
aditya@aditya-pc:~/temp/cpp/temp$ ./t
I am in pp
```