UNIT IV - TEMPLATES AND EXCEPTION HANDLING

- **Topics to be discussed,**
  - ➢Function Template and Class Template
  - ➢**Namespaces**
  - ➢Casting
  - ➢Exception Handling

Namespaces in C++

- Consider a situation, when we have two persons with the same name, Vijay, in the same class.
- Whenever we need to differentiate them definitely we would have to use some additional information along with their name, like either the area, if they live in different area or their mother’s or father’s name, etc.

Namespaces in C++ - Cont’d

- Same situation can arise in our C++ applications.
- For example, we might be writing some code that has a function called calc() and there is another library available which is also having same function calc().
- Now the compiler has no way of knowing which version of calc() function we are referring to within our code.

Namespaces in C++ - Cont’d

- A **namespace** is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc.
- with the same name available in different libraries, using namespace, we can define the context in which names are defined.
- In essence, a namespace defines a scope.

Namespaces in C++ - Cont’d

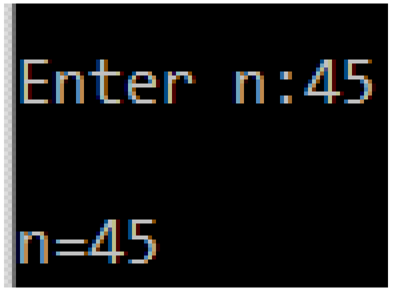
- A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it.
- Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when our code base includes multiple libraries.
- All identifiers at namespace scope are visible to one another without qualification.
- Identifiers outside the namespace can access the members by using the fully qualified name for each identifier

Namespaces in C++ - Cont'd

Example:

```
#include<iostream>
int main()
{
   int n;
   std::cout<<"\nEnter n:";
   std::cin>>n;
   std::cout<<"\nn="<<n;
}
```

Output:



```
Enter n:45

n=45
```

- Notice that we used, std::cin and std::cout instead of cin and cout
- The prefix std:: indicates that the names cin and cout are defined inside the namespace std.

Namespaces in C++ - Cont'd

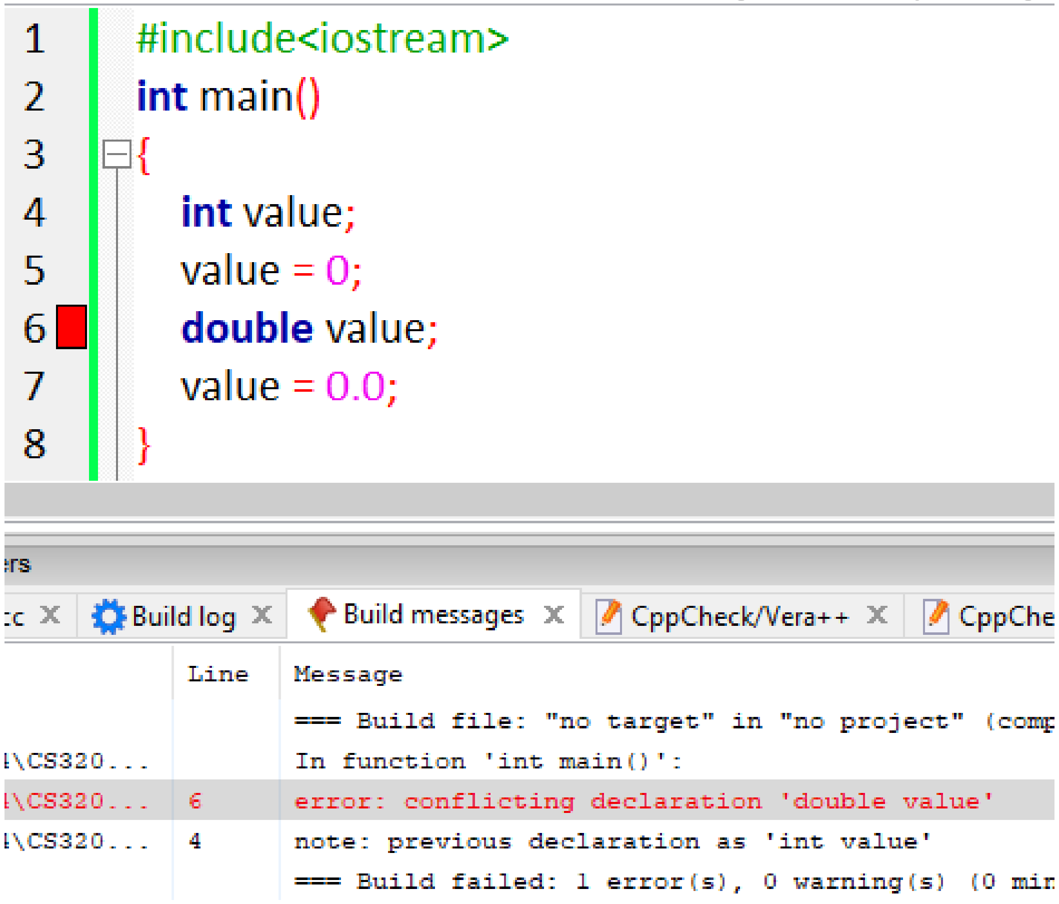
- Defining a Namespace
  - A namespace definition begins with the keyword **namespace** followed by the namespace name as follows,

```
namespace namespace_name
 {
// code declarations i.e. variable (int a;)
method (void add();)
classes ( class student{};)
}
```

- *It is to be noted that, there is no semicolon (;) after the closing brace*.
- To call the namespace-enabled version of either function or variable, prepend the namespace name as follows:
  - namespace_name: :code; // code could be variable , function or class.

Namespaces in C++ - Cont'd

- Consider the following C++ program:



```
1  #include<iostream>
2  int main()
3  {
4      int value;
5      value = 0;
6      double value;
7      value = 0.0;
8  }
```

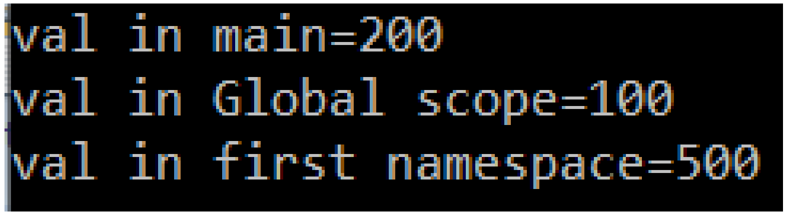
```
              Line   Message
                     === Build file: "no target" in "no project" (comp
:\CS320...           In function 'int main()':
:\CS320...    6      error: conflicting declaration 'double value'
:\CS320...    4      note: previous declaration as 'int value'
                     === Build failed: 1 error(s), 0 warning(s) (0 mir
```

- In each scope, a name can only represent one entity.
- So, there cannot be two variables with the same name in the same scope.
- Using namespaces, we can create two variables or member functions having the same name.

variables with the same name in the different scope - Example

```cpp
#include <iostream>
using namespace std;
// Variable created inside namespace
namespace first
{
        int val = 500;
}
 // Global variable
int val = 100;
int main()
{
    // Local variable
    int val = 200;
    cout<<"val in main="<<val<<endl;
    cout<<"val in Global scope="<<::val<<endl;
    cout << "val in first namespace="<<first::val << endl;
    return 0;
}
```

Output:

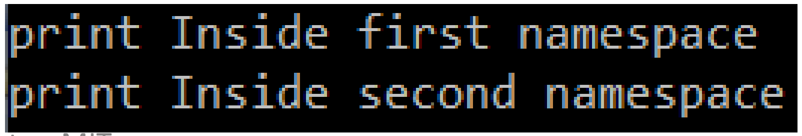


```
val in main=200
val in Global scope=100
val in first namespace=500
```

functions with the same name in the different scope - Example

```
#include <iostream>
using namespace std;
namespace first // first name space
{
  void print() {
    cout << "print Inside first namespace" << endl;
  }
}
namespace second // second name space
{
  void print() {
    cout << "print Inside second namespace" << endl;
  }
}
int main ()
 {
  // Calls function from first name space.
  first::print();
  // Calls function from second name space.
  second::print();
  return 0;
}
```

Output:



```
print Inside first namespace
print Inside second namespace
```

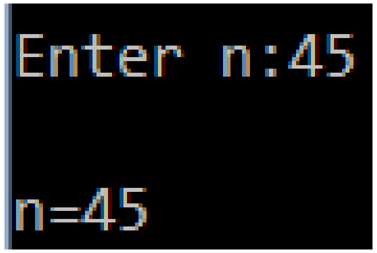
Namespaces in C++ - Cont'd

- **The using directive:**
  - We can avoid prepending of namespaces with the **using namespace** directive.
  - This directive tells the compiler that the subsequent code is making use of names in the specified namespace.

```
#include<iostream>
using std::cout;
int main()
{
    int n;
    cout<<"\nEnter n:";
    std::cin>>n;
    cout<<"\nn="<<n;
}
```

```
#include<iostream>
using namespace std;
int main()
{
    int n;
    cout<<"\nEnter n:";
    cin>>n;
    cout<<"\nn="<<n;
}
```

Output:

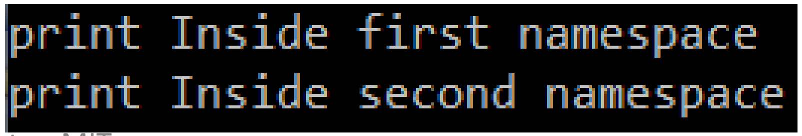


```
Enter n:45

n=45
```

```
#include <iostream>
using namespace std;
namespace first // first name space
{
  void print() {
    cout << "print Inside first namespace" << endl;
  }
}
namespace second // second name space
{
  void print() {
    cout << "print Inside second namespace" << endl;
  }
}
using namespace first;
int main ()
 {
  // Calls function from first name space.
  print();
  // Calls function from second name space.
  second::print();
  return 0;
}
```

Output:



```
print Inside first namespace
print Inside second namespace
```

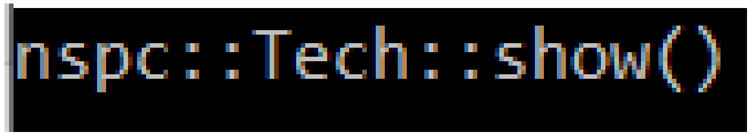
Namespaces in C++ - Cont'd

- **Classes and Namespace**
- As like variables and functions, class can also be defined inside a namespace

Classes in a Namespace - Example

```
#include<iostream>
using namespace std;
 namespace nspc
{
    class Tech // A Class in a namespace
    {
    public:
        void show()
        {
            cout<<"nspc::Tech::show()"<<endl;
        }
    };
}
int main()
{
    nspc::Tech obj;
    obj.show();
    return 0;
}
```

Output:

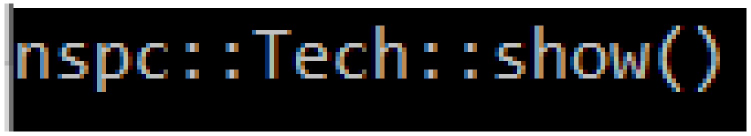


```
nspc::Tech::show()
```

A class can also be declared inside namespace and defined outside namespace

```
#include<iostream>
using namespace std;
 namespace nspc
{
    class Tech;
}
class nspc:: Tech
{
    public:
        void show()
        {
            cout<<"nspc::Tech::show()"<<endl;
        }
};
int main()
{
    nspc::Tech obj;
    obj.show();
    return 0;
}
```

Output:

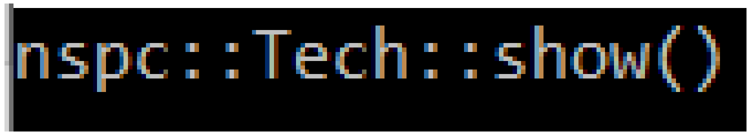


```
nspc::Tech::show()
```

We can define methods as well outside the namespace.

```
#include<iostream>
using namespace std;
 namespace nspc
{
    class Tech
    {
    public:
        void show();
    };
}
void nspc:: Tech:: show()
{
            cout<<"nspc::Tech::show()"<<endl;
}
int main()
{
    nspc::Tech obj;
    obj.show();
    return 0;
}
```

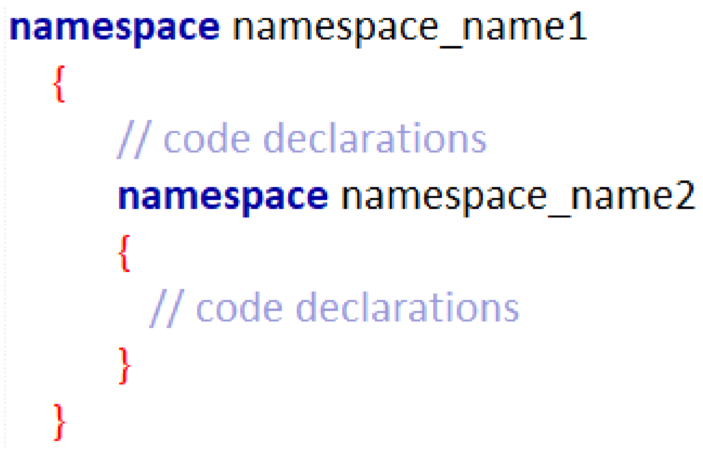
Output:



```
nspc::Tech::show()
```

Namespaces in C++ - Cont'd

- **Nested Namespaces:**
  - Namespaces can be nested where we can define one namespace inside another name space as follows:



```
namespace namespace_name1
  {
       // code declarations
       namespace namespace_name2
       {
         // code declarations
       }
  }
```

•We can access members of nested namespace by using resolution operators as follows:

// to access members of namespace_name2

using namespace namespace_name1::namespace_name2;

// to access members of namespace_name1

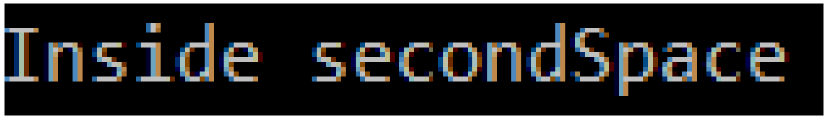
using namespace namespace_name1;

Nested Namespaces - Example

```cpp
#include<iostream>
using namespace std;
namespace firstSpace
{
   void func()
   {
      cout << "Inside firstSpace" << endl;
   }
  namespace secondSpace
   {
      void func()
      {
         cout << "Inside secondSpace" << endl;
      }
   }
}
```

```cpp
using namespace firstSpace::secondSpace;
int main ()
{
   // This calls function from second name space.
    func();
    return 0;
}
```

Output:



```
Inside secondSpace
```

Namespaces in C++ - Cont’d

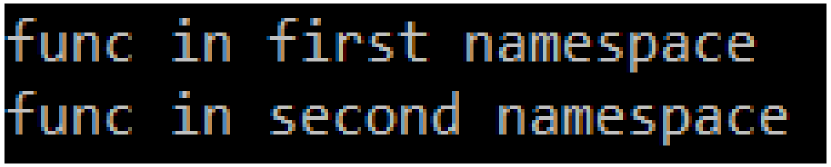
- **namespace extension :**
  - It is also possible to create more than one namespaces in the global space.
  - This can be done in two ways.
    - **namespaces having different names**
    - **Extending namespaces (Using same name twice)**

namespaces having different names - Example

```
#include <iostream>
using namespace std;
namespace first
{
  void func()
  {
     cout<<"func in first namespace\n";
   }
}
namespace second
{
  void func()
  {
     cout<<"func in second namespace\n";
   }
}
```

```
int main()
{
  first::func() ;
  second::func() ;
  return 0;
}
```

Output:



```
func in first namespace
func in second namespace
```

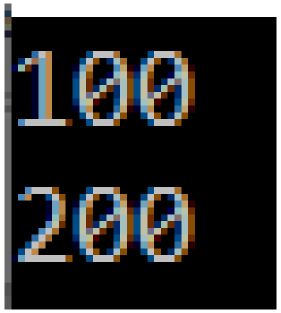
Namespaces in C++ - Cont’d

- **Extending namespaces (Using same name twice)**
  - It is also possible to create two namespace blocks having the same name.
  - The second namespace block is nothing but actually the continuation of the first namespace.
  - In simpler words, we can say that both the namespaces are not different but actually the same, which are being defined in parts.

Extending namespaces (Using same name twice) - Example

```
#include <iostream>
using namespace std;
namespace first
{
   int n1 = 100;
}
namespace  first
{
   int n2 = 200;
}
int main()
{
   cout << first::n1 <<"\n";
   cout << first::n2 <<"\n";
   return 0;
}
```

Output:



```
100
200
```

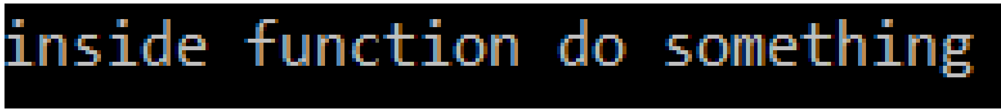
Namespaces in C++ - Cont'd

- **Unnamed Namespaces**
  - They are directly usable in the same program and are used for declaring unique identifiers.
  - In unnamed namespaces, name of the namespace is not mentioned in the declaration of namespace.
  - The name of the namespace is uniquely generated by the compiler.
  - **The unnamed namespaces we have created will only be accessible within the file we created it in.**
  - Unnamed namespaces are the replacement for the static declaration of variables.

Unnamed Namespaces - Example

```
#include <iostream>
using namespace std;
namespace // unnamed namespace
{
    void doSomething() // can only be accessed in this file
    {
        cout << "inside function do something\n";
    }
}
int main()
{
    doSomething(); // we can call doSomething() without a namespace prefix
    return 0;
}
```

Output:



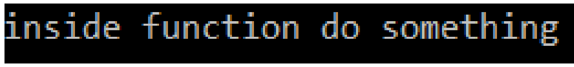
```
inside function do something
```

```cpp
#include <iostream>
using namespace std;
namespace // unnamed namespace
{
    void doSomething() // can only be accessed in this file
    {
        cout << "inside function do something\n";
    }
}
int main()
{
    doSomething(); // we can call doSomething() without a namespace prefix
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
static void doSomething() // can only be accessed in this file
{
    cout << "inside function do something\n";
}
int main()
{
    doSomething(); // we can call doSomething() without a namespace prefix
    return 0;
}
```

Both are same

Output:



```
inside function do something
```