# Header Files

# C++: Namespace

- **Namespace**
  - Similar to “import” in Java
    - Ex. List<Integer> numbers = new ArrayList<Integer>();
    - Requires Import javax.util.List and javax.util.ArrayList
  - Similarly,
    - Using namespace std;
      - Allows you to use “commands” such as cin, cout, etc.

# C++: Namespace

- **Namespace**
  - Global Namespace
    - If no namespace is used, all the code is in “global namespace”
    - No need to use “using” directive
  - To define your own namespace
    - Use “namespace *my_name_space*”
      - Now the code will be in this namespace

# C++: Namespace

- **Example**
  - » Library1 namespace has method getArea and getSurfaceArea
  - » Library11 namespace within Library1 has a method getSurfaceArea

```cpp
#include <iostream>

namespace Library1{

  int getArea(int width, int height){
    return width*height;
  }

  int getSurfaceArea(int width, int height, int length){
    return 2*width*height*length;
  }

  namespace Library11{

    int getSurfaceArea(int width, int height, int length){
      return 2*width*height*length;
    }
  }

}
```

# C++: Namespace

## — Example

» To use the namespace,

**using namespace** <<namespace>>

» Library11 namespace within Library1 has a method getSurfaceArea

```
using namespace Library1;

int main(){

  function();
  cout<<"Area: "<<Library1::getArea(10,20)<<endl;
  cout<<"Area: "<<Library11::getSurfaceArea(20,2,2)<<endl;
}
```

# C++: Header Files

- **Why ?**
  - Previously, in a single C++ file (.cpp), we have:
    - Forward declared the function the program is going to define *(interface)*
    - Later, define the function to implement the functionality *(implementation)*

# C++: Header Files

- **Problem**
  - » If the class is simple, mixing *declaration* and *implementation* is easy to manage
  - » If **NOT**, its not <u>good practice</u> to write both *declaration* and *implementation* in the same file
  - » Also, mixing *declaration* and *implementation* only allows one implementation.
    - › Same file has to be modified for alternative implementation

# C++: Header Files

— **Example**

```
class Teacher{

private:
            int SSN = 007;
protected:
            string taxID = "MI5007";
public:
      string firstName;
      string lastName;
      string email;
Teacher(string fname,string lname,string email){
      this->firstName = fname;
      lastName = lname;
      this->email = email;
}
Teacher(const Teacher& obj){
      firstName = obj.firstName;
      lastName = obj.lastName;
      email = "rks-"+obj.email;
}
void sendEmail() const{
            cout <<"Test Email is sent to "<<getEmail()<<endl;
}
string getEmail() const{
            return email;
}
};
```

Declaration

Definition
or
implementation

# C++: Header Files

— **Example**

Declaration

```
class Teacher{

private:
            int SSN = 007;
protected:
            string taxID = "MI5007";
public:
      string firstName;
      string lastName;
      string email;
Teacher(string fname,string lname,string email){

}

Teacher(const Teacher& obj){

}
void sendEmail() const{

}
string getEmail() const{

}
};
```

# C++: Header Files

Refer to Teacher.h

```
#include <iostream>
using std::string;
#ifndef TEACHER_H_
#define TEACHER_H_

class Teacher {
        private:
                string SSN = “007-007-0007”;
        protected:
                string taxID = "MI5007";
        public:
                string firstName;
                string lastName;
                string email;
                Teacher(string fname,string lname,string
                email);
                Teacher(const Teacher& obj);
                void sendEmail() const;
                string getEmail() const;
};
#endif /* TEACHER_H_ */
```

**Declaration of Teacher class**

**Teacher.h**

# C++: Header Files

## Implementation of Teacher Teacher.cpp

— **Example**

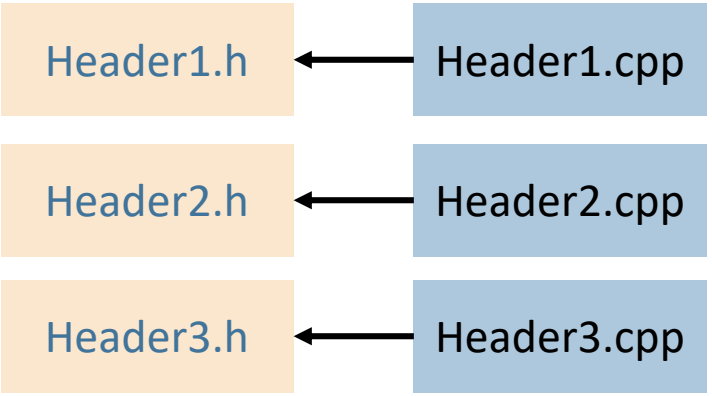
```
#include "Teacher.h"

Teacher::Teacher(string fname,string lname,string email){
    this->firstName = fname;
    lastName = lname;
    this->email = email;
}
Teacher::Teacher(const Teacher& obj){
    firstName = obj.firstName;
    lastName = obj.lastName;
    email = "rks-"+obj.email;
}
void Teacher::sendEmail() const{
        cout <<"Test Email is sent to "<<getEmail()<<endl;
}
string Teacher::getEmail() const{
        return email;
}
};
```

# C++: Header Files

- **Header Files**
  - Captures a *class* declaration
  - Acts as an *interface* for a class or function implementation
  - Always include a header file “*.h” NOT its implementation (*.cpp)

# How does it work ?




- **At the compile time**
  - Preprocessing – the **#include** directive is executed
    - Header file (.h) contents are included into the source file (.cpp)
    - *source files* are PASSED to the compiler
  - Compilation – the source files are converted into respective object files (.o)
  - Linker – the linker brings the object files together to generate the executable output

# C++: Header Files

— **Guidelines**

» Keep a module's or .cpp internal declarations out of the header file

› Something specific to the implementation

› How to check - Header file content should compile correctly by itself

» Every header file should #include every other header file that the current header file requires to compile correctly

» Avoid #include a .cpp file for any reason

# C++: Header Files

— **Guidelines**

» Each module with its **.h** and .cpp file should correspond to a clear piece of functionality.

› Header file contains *declarations*, *function prototypes*, and *global variable extern declarations*

› Source file contains *function definitions*, *global variable definitions* and *initializations*

» Always use "include guards" in a header file

```
#ifndef EXAMPLEHEADER_H_
#define EXAMPLEHEADER_H_
```

# Thank You

Question, Comments & Feedback