Week 9 Lecture 1

NWEN 241
Systems Programming

Jyoti Sahni

jyoti.sahni@ecs.vuw.ac.nz

Content

- Introduction to C++
- Data Types
- Input Output

Announcement

- Assignment 4 is out.
  - Submit by May 30, 2022 23:59

- Exercise 4 will be released tomorrow
  - Submit by May 20, 2022 23:59

Introduction to C++

- Developed by Bjarne Stroustrup in 1979.
- Objective was to develop efficient and flexible language similar to C that also provided high-level features for program organization
- C++ used to be initially called "C with Classes"
- The first standard appeared in 1998 - C++98.

Introduction to C++

- Latest standard - C++ 20.
  - Over time, both C and C++ evolved and have diverged from one another.
  - Compatibility between the languages is has always been considered important.

- Widely use for video games, embedded systems, IoT and resource-heavy VR and AI applications

Introduction to C++

- C++ extends C

- Is C++ a superset of C ?
    - **Not anymore**

- Not **every** C program can run on a C++ complier

- **Most** of the C programs run on a C++ compiler

Introduction to C++

C++ extends C with

- strong type checking
- new data types
- **Classes** for implementing user defined data types
- **object-oriented programming** (also supports - procedural and functional).
- a **standard library** (STL) for frequently used data types (string, list, stack, queue, vector, hash,…)
- generic programming, i.e., parameterization of variable types via **templates**
- exceptions for error handling

In this course we focus on OOP using C++

Comparing C, C++ and Java

C is the basis for C++ and Java

- C evolved into C++
- C++ transmuted into Java
- The “class” is an extension of “struct” in C

Similarities

- Java uses a syntax similar to C and C++ (for, while, …)
- Java supports OOP as C++ does (class, inheritance, …)

Differences

- Everything extends into an Object class in Java (You cannot code anything in Java without declaring classes and objects.); not everything is an object in C or C++
- Java does not support pointer
- Java frees memory by garbage collection
- Java is more portable by using bytecode and virtual machine
- Java does not support operator overloading
- …

Comparing C++ and Java

Java: Compiled and interpreted

C++: Compiled (same as C)

Java: Everything must be placed in a class.

C++: We can define functions and variables outside a class, using the same syntax used in C. The “main” function must be defined outside a class.

Java: All user-defined types are classes.

C++: We can define C types (enum, struct, array).

Java: No explicit pointers.

C++: Explicit pointers as well as references are available.

Comparing C++ and Java

Java: All objects are allocated on the heap (dynamically allocated using new keyword).

C++: An object can be allocated on the heap or on the stack.

Java: Single Inheritance

C++: Multiple Inheritance

Java: Automatic memory management.

C++: Manual memory management (like C) or semiautomatic management (shared pointers – will learn about this later)

Moving from C to C++

hello.cpp

```
#include<stdio.h>
int main()
{
   printf("Hello World");
}
```

Programs written in C may be valid in C++

Compile: g++ hello.cpp –o helloex

Run: ./helloex

Moving from C to C++

- **Headers:**
  - In standard C++, library headers are **not** supposed to have an extension.
  - C++ standard neither requires nor forbids an extension for other headers. The common choices of extension of user defined headers are- ***.h / *.hh /* .hpp**
- **C compatibility headers**
  - For some of the C standard library headers of the form xxx.h, the C++ standard library both includes an identically-named header and another header of the form cxxx.

Moving from C to C++

hello.cpp

```
#include<cstdio>
int main()
{
   printf("Hello World");
}
```

Compile: g++ hello.cpp –o helloex

Run: ./hello

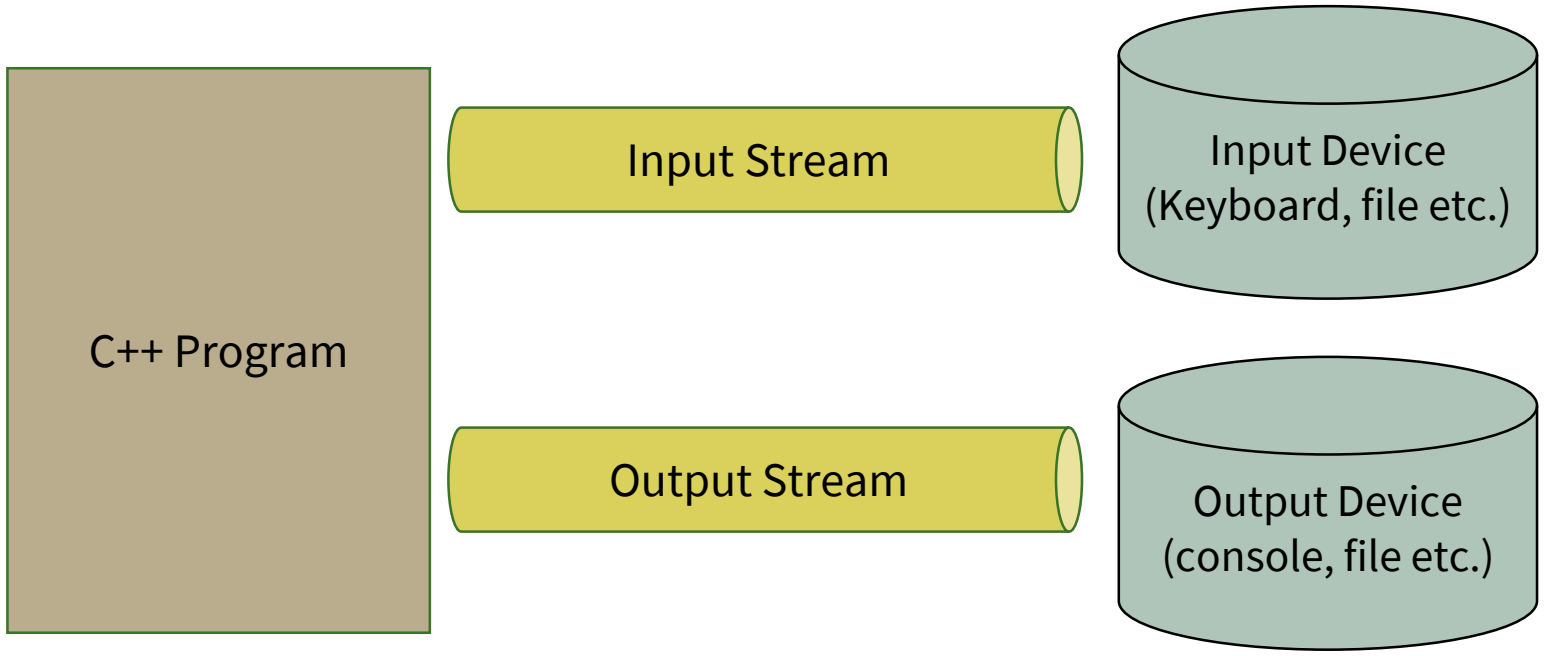
Primitive Data Types

Data Type	Keyword	Modifiers
Character	char	signed, unsigned
Integer	int	signed, unsigned, short, long, long long
Float	float	
Double	double	long
Boolean	bool	
Wide character representations	wchar_t	

I / O in C++

I/O Basics

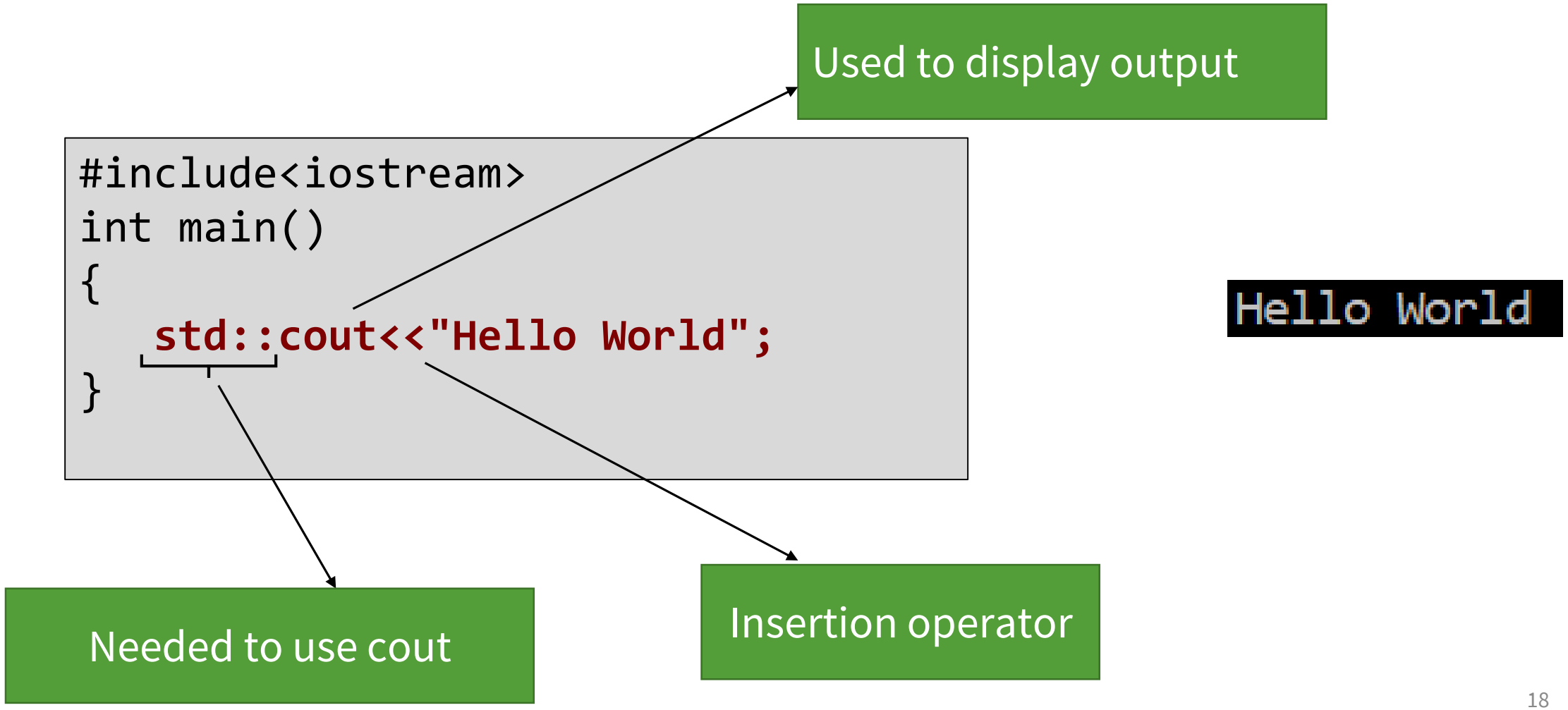
- C/C++ I / O are based on *streams*, which are sequence of bytes flowing in and out of the programs
- Streams acts as an intermediaries between the programs and the actual I/O devices



With **input** streams, the **extraction operator (>>)** is used to remove values from the stream

With **output** streams, the **insertion operator (<<)** is used to put values in the stream

Standard output



Standard Input

```
#include<iostream>
int main()
 {
   int i;
   std::cin>>i;

   std::cout<<"You entered"<<i;
 }
```

Used to read input

extraction operator

```
125
You entered 125
```

Needed to use cint

Namespace

- **Namespaces** are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.
- General syntax:

```
namespace namespace_name
{
    members
}
```

- Members can be constants, variables, functions, classes, or another namespace (nested namespaces)

Example:

```
namespace myns
{
   const int N = 100;
   int count = 0;
   void printResult();
}
```

Namespace

- The scope of a namespace member is **local** to that namespace. All identifiers at namespace scope are visible to one another without qualification.

- Members are **not** visible **outside** its namespace.

- Everything not declared in another namespace is in the global (program-wide) namespace.

Two ways to access a namespace member outside its namespace:

- Use `namespace_name::identifier` syntax
- Use the `using` keyword to access specific or all members of a namespace

Example

```
namespace myns
{
    const int N = 100;
    int count = 0;
    void printResult();
}
```

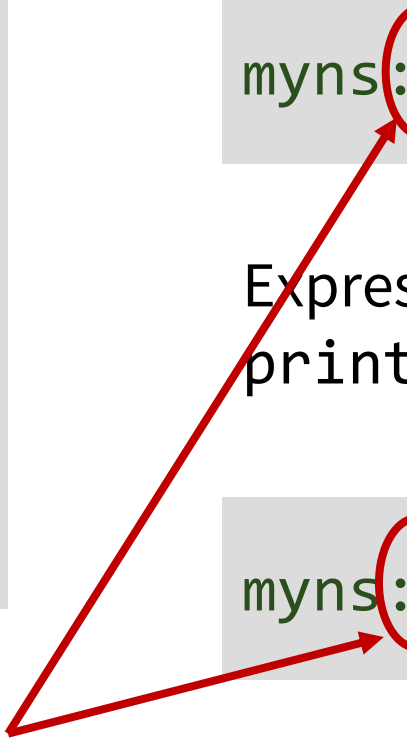
Expression to access N:

```
myns::N
```

Expression to invoke `printResult()`:

```
myns::printResult();
```

Scope resolution operator



Example

```
namespace myns
{
    const int N = 100;
    int count = 0;
    void printResult();
}
```

Make all members visible to the current scope

```
using namespace myns;
```

Expression to access N:

```
N
```

Expression to invoke `printResult()`:

```
printResult();
```

Example

```
namespace myns
{
    const int N = 100;
    int count = 0;
    void printResult();
}
```

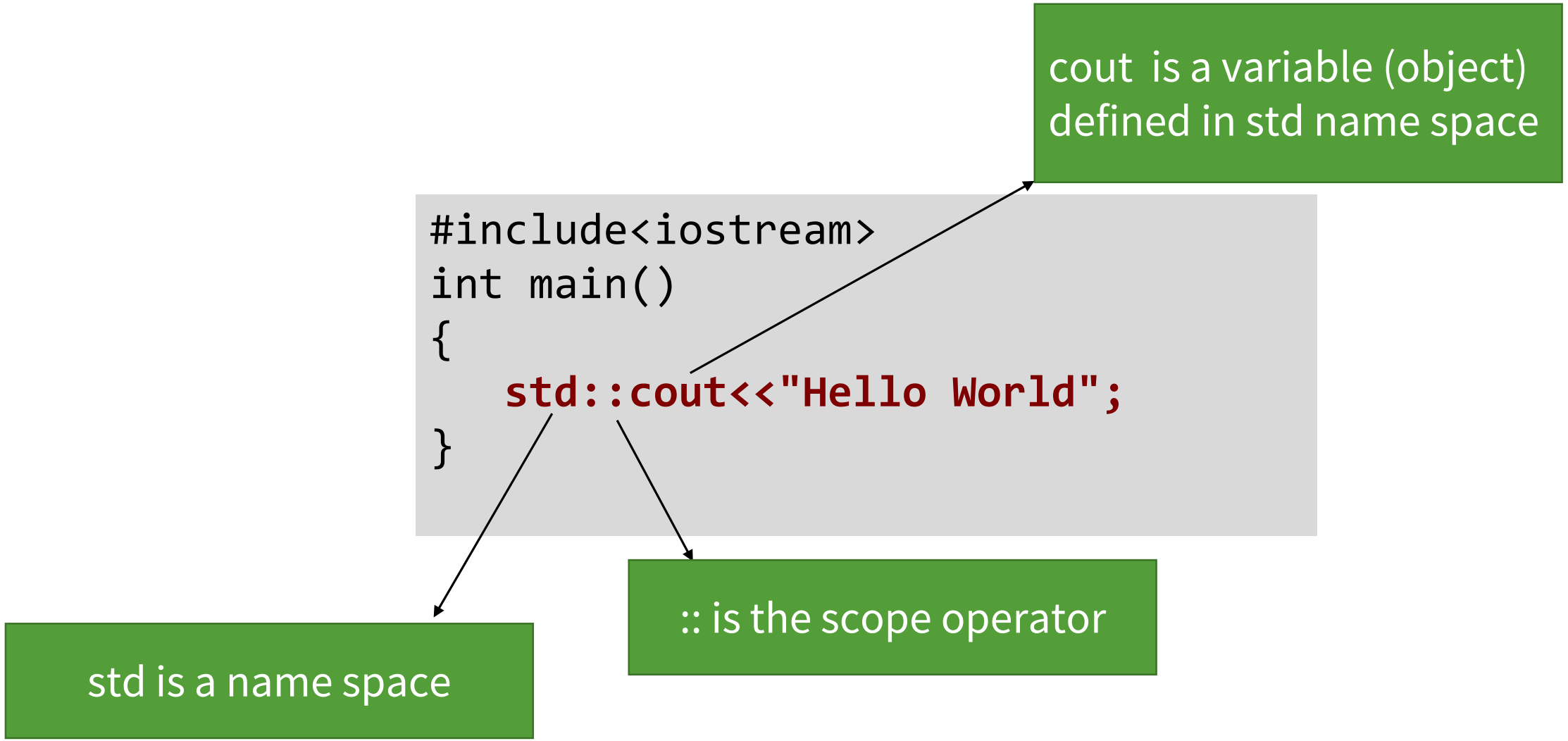
Make a specific member visible

```
using myns::N;
```

Expression to access N:

```
N
```

Revisit – Standard Output



Revisit – Standard Output

cout is a variable (object) defined in std name space

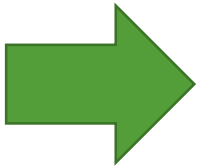
```
#include<iostream>
using namespace std;
int main()
{
   cout<<"Hello World";
}
```

Example

```
#include <iostream>

int main(void)
{
   std::cout << "Hello world\n";

   return 0;
}
```



```
#include <iostream>

using namespace std;

int main(void)
{
   cout << "Hello world\n";

   return 0;
}
```

Standard I/O in C++ - The complete picture

- C++ has a number of classes for defining various streams related with standard input-output operations.
- These classes are defined in **iostream** header file.
- C++ comes with four predefined standard stream objects

Stream	Description	Remarks
cin	Tied to the standard input	Typically tied to the keyboard
cout	Tied to the standard output	Typically tied to the monitor
cerr	Tied to the standard error, providing unbuffered* output	Typically tied to the monitor
clog	Tied to the standard error, providing buffered* output	Typically tied to the monitor

* Unbuffered output is typically handled immediately, where as buffered output is typically stored

Program Structure

- A typical C++ program consists of
  - 1 or more **header** files
  - 1 or more C++ **source** files

```
#include <iostream>

int main(void)
{
   std::cout << "Hello world\n";

   return 0;
}
```

Preprocessor directive to include `iostream` header file which contains `std::cout`

`main` function *definition*, invoking `std::cout` to display “Hello, world”, and return 0

Another example program (C++)

```
/* Program to calculate the area of a circle */
#include <iostream>
#define PI 3.14

float sq(float);

int main(void)
{
   float radius, area;

   /* Ask user to input */
   std::out << "Radius = ";
   std::in >> radius;

   area = PI * sq(radius);
   std::out << "Area = " << area << "\n";
   return 0;
}

float sq(float r)
{
   return (r * r);
}
```

Preprocessor directives

Function prototype

main function

Function definition

Next Lecture

- Classes in C++