Week# 1
Introduction to Computers and Programming

Dr Taimur Ahmed

Lecture# 2
Basic of Typical C++ Environment, Good Programming Practice

Introduction to Programming

- ❑ What is a **Program**?
  - ➢ A program is **a precise sequence of steps to solve a particular problem**
  - ➢ A complete set of activities to be performed in a particular order – what’s the purpose?

- ❑ What is **Programming**?
  - ➢ A process of **designing and building an executable computer program to accomplish a specific computing result or to perform a specific task**
  - ➢ e.g solve some mathematical problems, video game for fun, edit images/videos, an app to browse internet or send/receive emails/messages, etc

Basics of a Typical C/C++ Environment

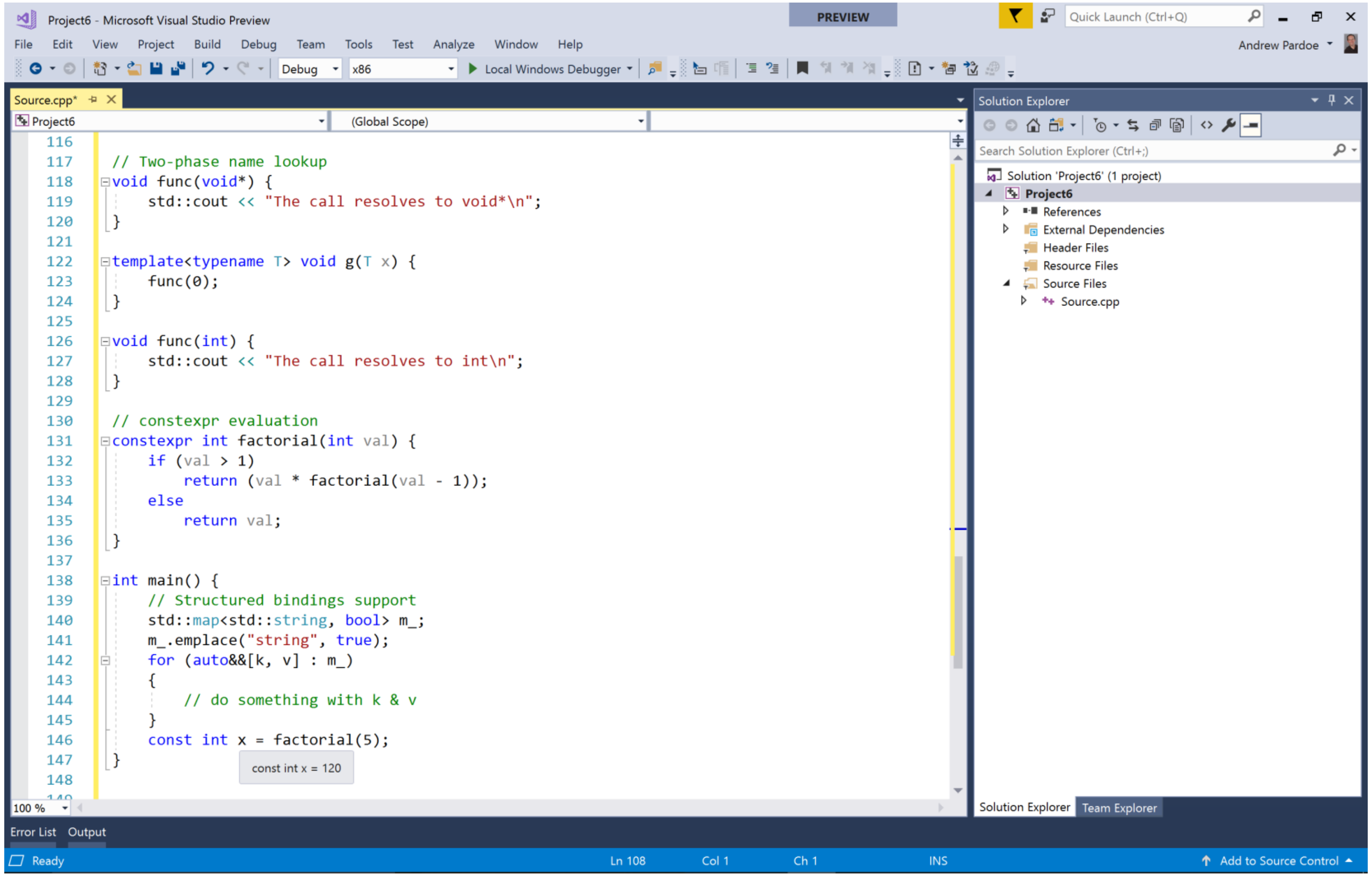
- C++ Language definition
- Program-development environment (tools)
  - compiler, linker, editor, debugger
- C++ Standard Library (software)
  - precompiled routines you can use

Programming IDE

- What is an IDE?
  - **Integrated Development Environment (IDE):** **A software that provides a user-friendly environment** for building applications that combines common developer tools into a single **graphical user interface** (GUI).
- An IDE typically consists of:
  - **Source code editor:** A text editor that can assist in writing code with features such as syntax highlighting with visual clues, providing language specific auto-completion, and checking for bugs as code is being written.
  - **Debugger:** A program for testing other programs that can graphically display the location of a bug in the original code.
  - **Local build automation:** Utilities that automate simple, repeatable tasks as part of creating a local build of the software for use by the developer, like compiling computer source code into binary code, packaging binary code, and running automated tests.

IDE for C/C++

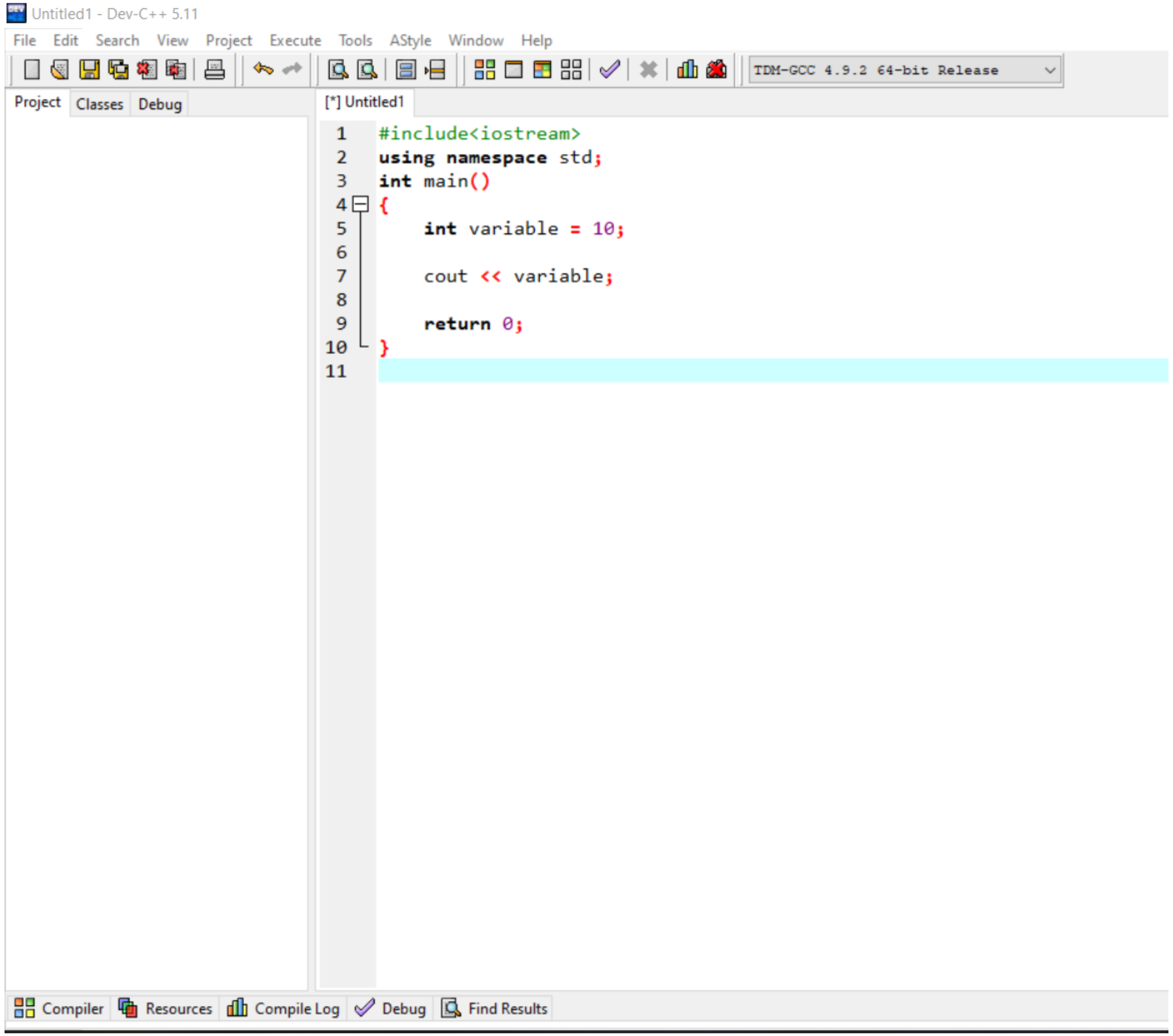
- Available IDEs for C/C++
  - Dev C++
  - Visual Studio Code
  - Eclipse
  - Net Beans
  - Code Blocks
  - There are **MANY MORE...**



```cpp

// Two-phase name lookup
void func(void*) {
    std::cout << "The call resolves to void*\n";
}

template<typename T> void g(T x) {
    func(0);
}

void func(int) {
    std::cout << "The call resolves to int\n";
}

// constexpr evaluation
constexpr int factorial(int val) {
    if (val > 1)
        return (val * factorial(val - 1));
    else
        return val;
}

int main() {
    // Structured bindings support
    std::map<std::string, bool> m_;
    m_.emplace("string", true);
    for (auto&&[k, v] : m_)
    {
        // do something with k & v
    }
    const int x = factorial(5);
}
```

const int x = 120

Dev C++ IDE

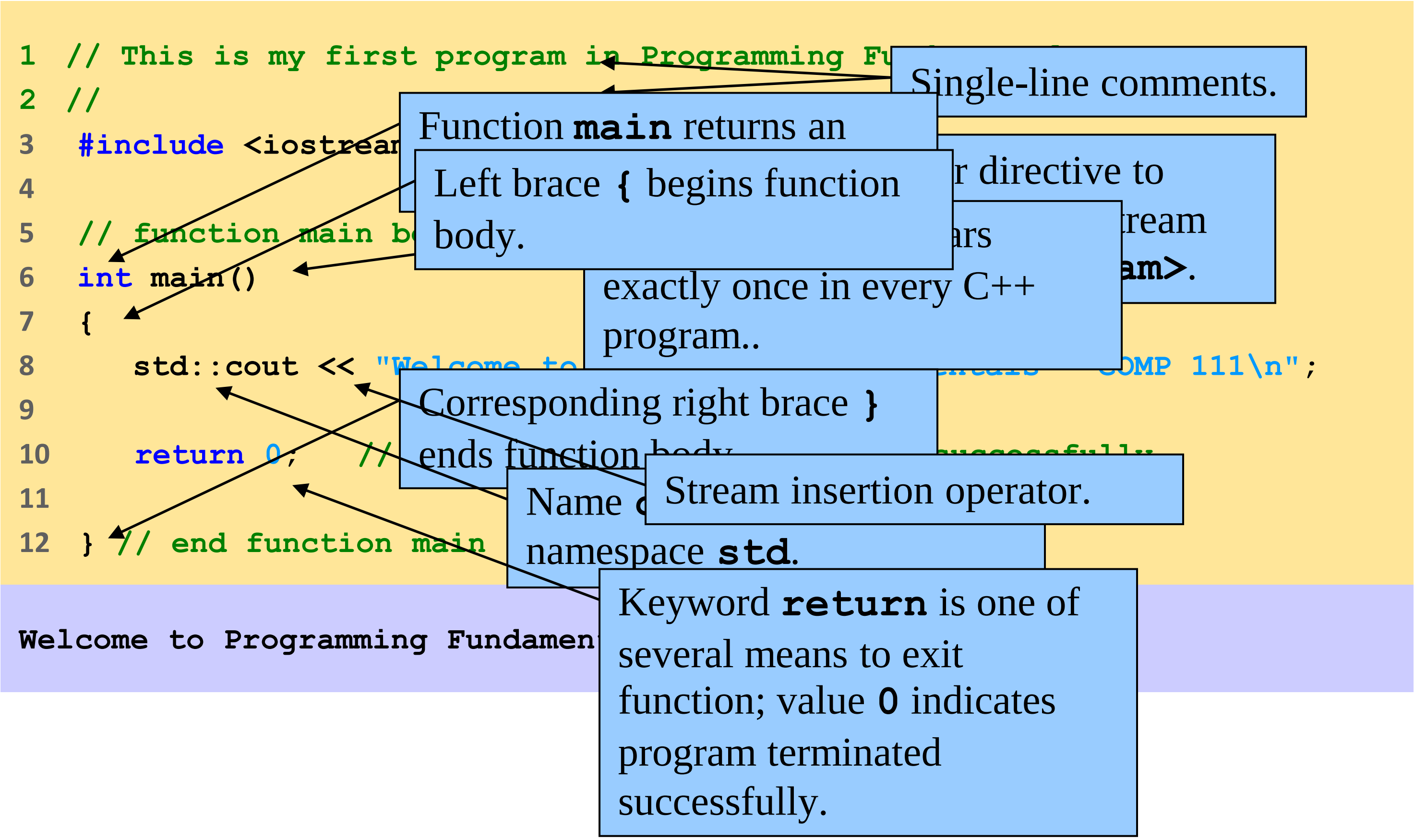


```
#include<iostream>
using namespace std;
int main()
{
    int variable = 10;

    cout << variable;

    return 0;
}

```

IDEs for C/C++

- ❑ Dev C++
  - ➢ Open source
  - ➢ Easy to use and setup
  - ➢ Simple full-featured IDE for C/C++ coding
  - ➢ Debugging feature
  - ➢ Syntax highlighting
  - ➢ Project/file manager
- ❑ Dev C++ installation
  - ➢ Download .exe file from https://sourceforge.net/projects/orwelldevcpp/

Lets Code!



```
// This is my first program in Programming Fu
//
 #include <iostrea

 // function main be
 int main()
 {
    std::cout << "Welcome to                        entals   COMP 111\n";

    return 0;    //                                  successfully

 } // end function main
```

```
Welcome to Programming Fundamen
```

Single-line comments.

Function **main** returns an

Left brace **{** begins function body.

r directive to

tream

am>.

ars

exactly once in every C++ program..

Corresponding right brace **}** ends function body.

Name o

namespace **std**.

Stream insertion operator.

Keyword **return** is one of several means to exit function; value **0** indicates program terminated successfully.

Lets Code!

Preprocessor directives

- Processed by preprocessor before compiling
- Begin with #

Input/output streams in C++

- **cin** **(pronounce "see in")**
  - Standard input stream
  - Normally keyboard
- **cout** **(pronounce "see out")**
  - Standard output stream
  - Normally computer screen

Comments

- Document your programs - **ALWAYS**
- Improve program readability
- Ignored by compiler
- Single-line comment
  - Begin with //

Lets Code!

- Standard **output** **stream object**
  - **std::cout**
  - “Connected” to screen
  - <<
    - Stream insertion operator
    - Value to right (right operand) inserted into output stream
- **Namespace**
  - **std::** specifies using name that belongs to “namespace” **std**
  - **std::** can be removed through use of following statements

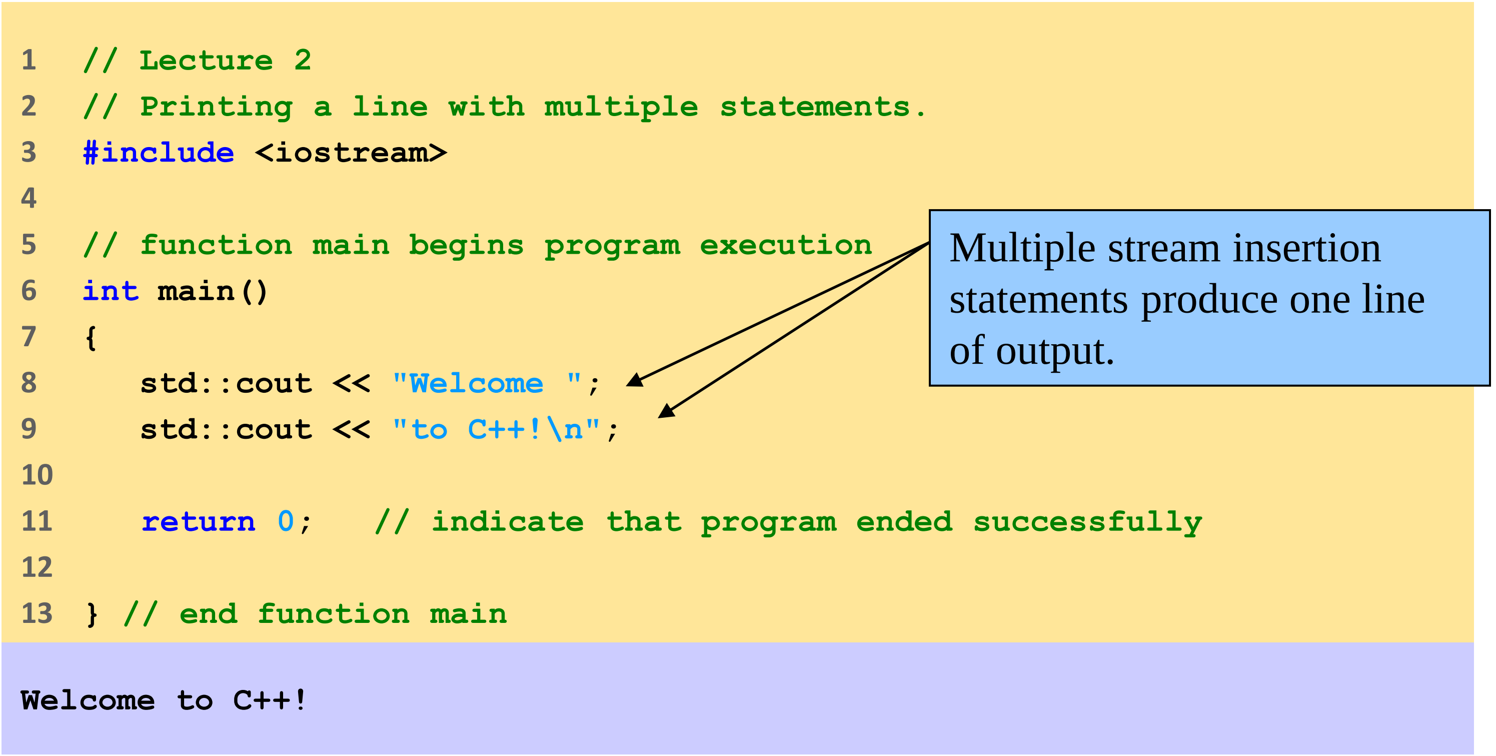
```
using namespace std;
```

Lets Code!

- **Escape** characters \
  - Indicates “special” character output

Escape Sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote character.

Lets Code!

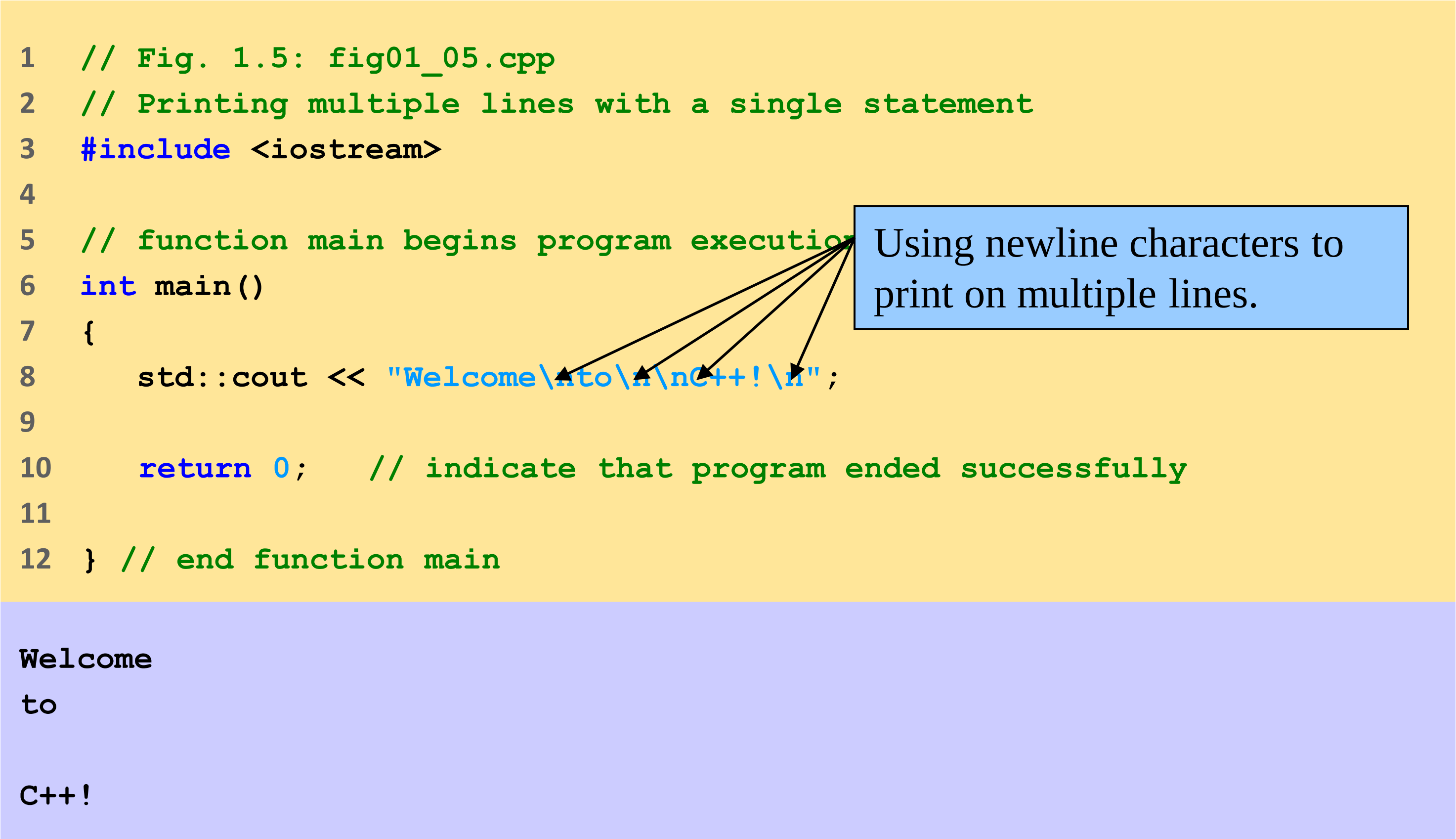


```
// Lecture 2
// Printing a line with multiple statements.
#include <iostream>

// function main begins program execution
int main()
{
   std::cout << "Welcome ";
   std::cout << "to C++!\n";

   return 0;   // indicate that program ended successfully

} // end function main
```

```
Welcome to C++!
```

Multiple stream insertion statements produce one line of output.

Lets Code!

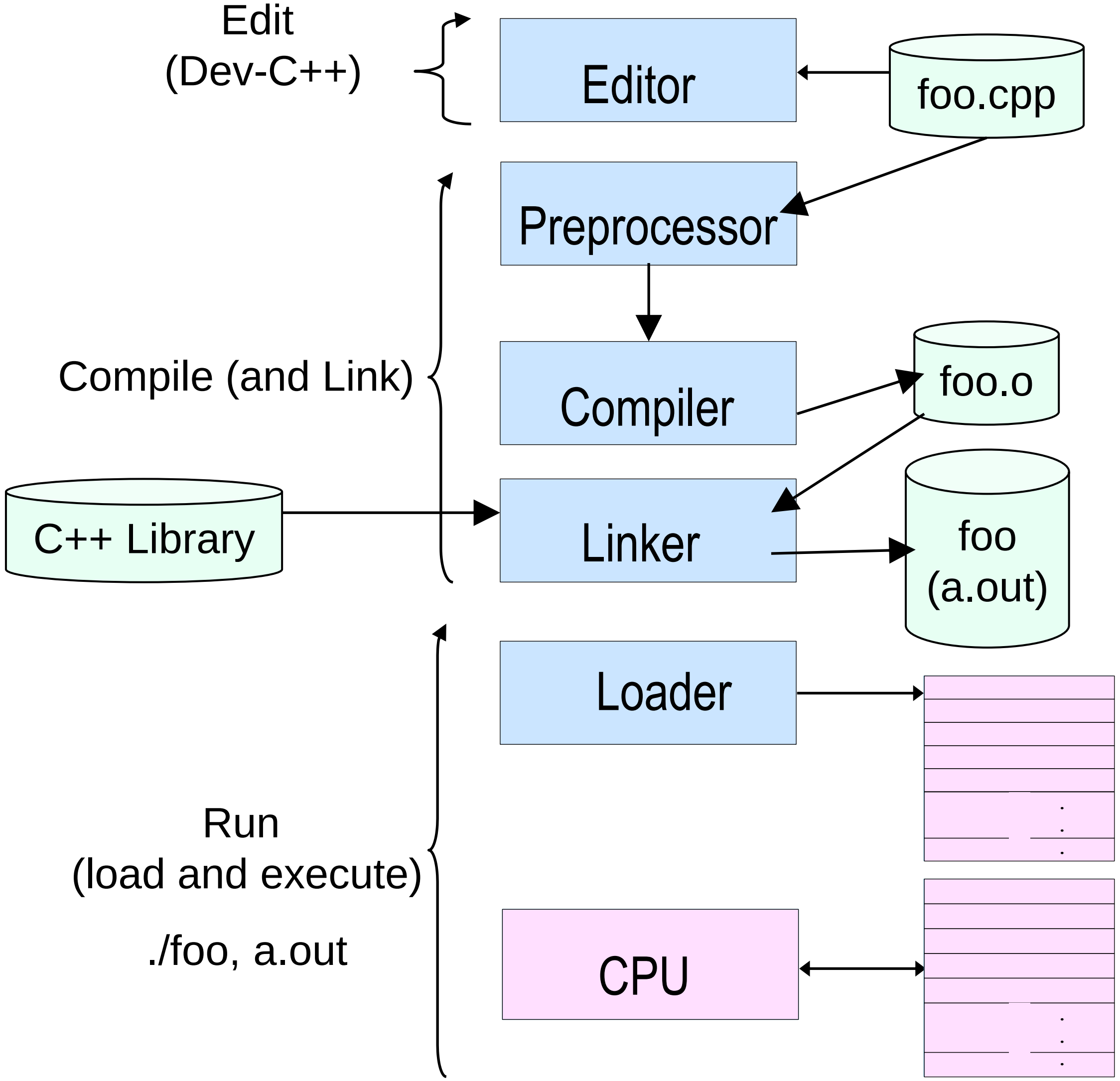


```cpp
// Fig. 1.5: fig01_05.cpp
// Printing multiple lines with a single statement
#include <iostream>

// function main begins program execution
int main()
{
   std::cout << "Welcome\nto\n\nC++!\n";

   return 0;   // indicate that program ended successfully

} // end function main
```

Using newline characters to print on multiple lines.

```
Welcome
to

C++!
```

C++ Code Execution Steps



Program created in the editor and stored on disk.

Preprocessor strips out comments, expands macros

Compiler creates object code and stores it on disk (.o file)

Linker links the object code with the libraries, stores in a .out on disk.

Loader puts program in memory (RAM)

CPU takes each instruction and executes it, possibly storing new data values in memory (RAM) as the program executes.

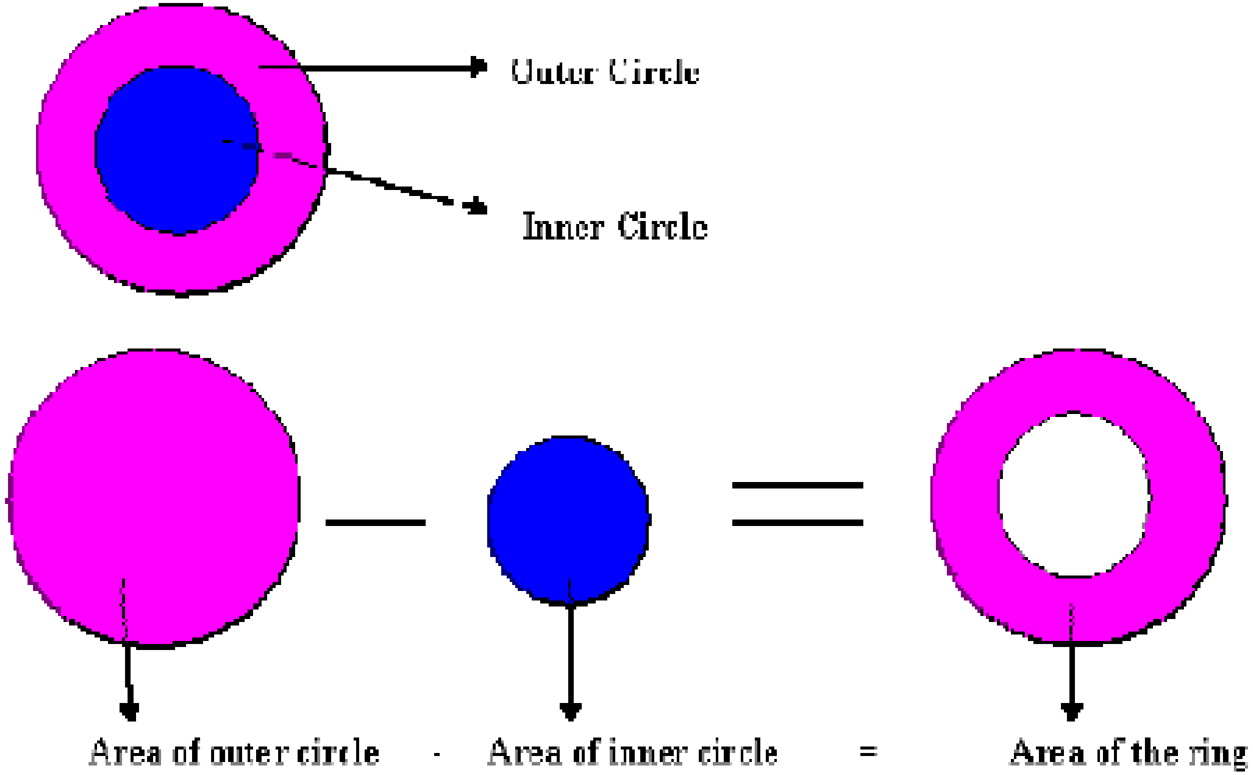
Good Programming Practices!

What you must always keep in mind while programming?

Skills required for Programming

- **Analytical Skills** - Paying **Attention** to Details
  - Fully understand the problem at hand
  - Pay attention to the logic e.g. *Ahmad sleeps 30 hours a day*
- **Reusability** of code
- Think about user and user interface
- Understand the fact that **computers are stupid**
- Comment the code (**always**)

Reusability of a Program



- ❑ While writing a program, always keep in mind that it could be reused at some other time
- ❑ Write in a way that it can be used to solve some other related problem
- ❑ Suppose we have to calculate the area of a given circle. We know the area of a circle is (**Pi * r^2**). Now we have written a program which calculates the area of a circle with given radius. At some later time we are given a problem to find out the area of a ring. The area of the ring can be calculated by subtracting the area of outer circle from the area of the inner circle. Hence, we can use the program that calculates the area of a circle to calculate the area of the ring.

Think about Good User Interfaces (UI)

- Never assume that users know a lot of things, this is a big mistake
- Never assume the user of your program as computer literate
- Try to design your UIs in a way to minimize the chances of errors/mistakes
- Always provide an easy to understand and easy to use interface that is self explanatory

Think of computers as stupid entity

- Computers do exactly what you tell them to do: no more, no less -- unlike human beings.
- Computers can't think by themselves. In this sense, they differ from human beings.
- For example, if someone asks you, “What is the time?”, “Time please?” or just, “Time?” you understand anyway that he is asking the time, but computer is different.
- Instructions to the computer should be explicitly stated. Computer will tell you the time only if you ask it in the way you have programmed it.
- When you're programming, it helps to be able to “think” as stupidly as the computer does, so that you are in the right frame of mind for specifying everything in minute detail, and not assuming that the right thing will happen by itself

Always Comment the Code

- ❑ **Always comment your code.**

- ❑ Comments are used to explain the **purpose** of the statements and programs. It helps the **other programmers** as well as the **creator** of the program to understand the code.

- ❑ The comment statements **do not affect the performance** of the program as these are ignored by the compiler and do not take any memory in the computer.