Programming Using C++

Lecture 10

Programming Using C++

- C++ is an extension of C, developed by Bjarne Stroustrup at Bell Labs during 1983-1985.
- C++ added a number of features that improved the C language. Most importantly, it added the object-oriented programming.
- C++ facilitates structured and disciplined approach to computer program design.
- C, C++, Java, and C# are very similar. Java was modeled after C++. C# is a subset of C++ with some features similar to Java. If you know one of these languages, it is easy to learn the others.

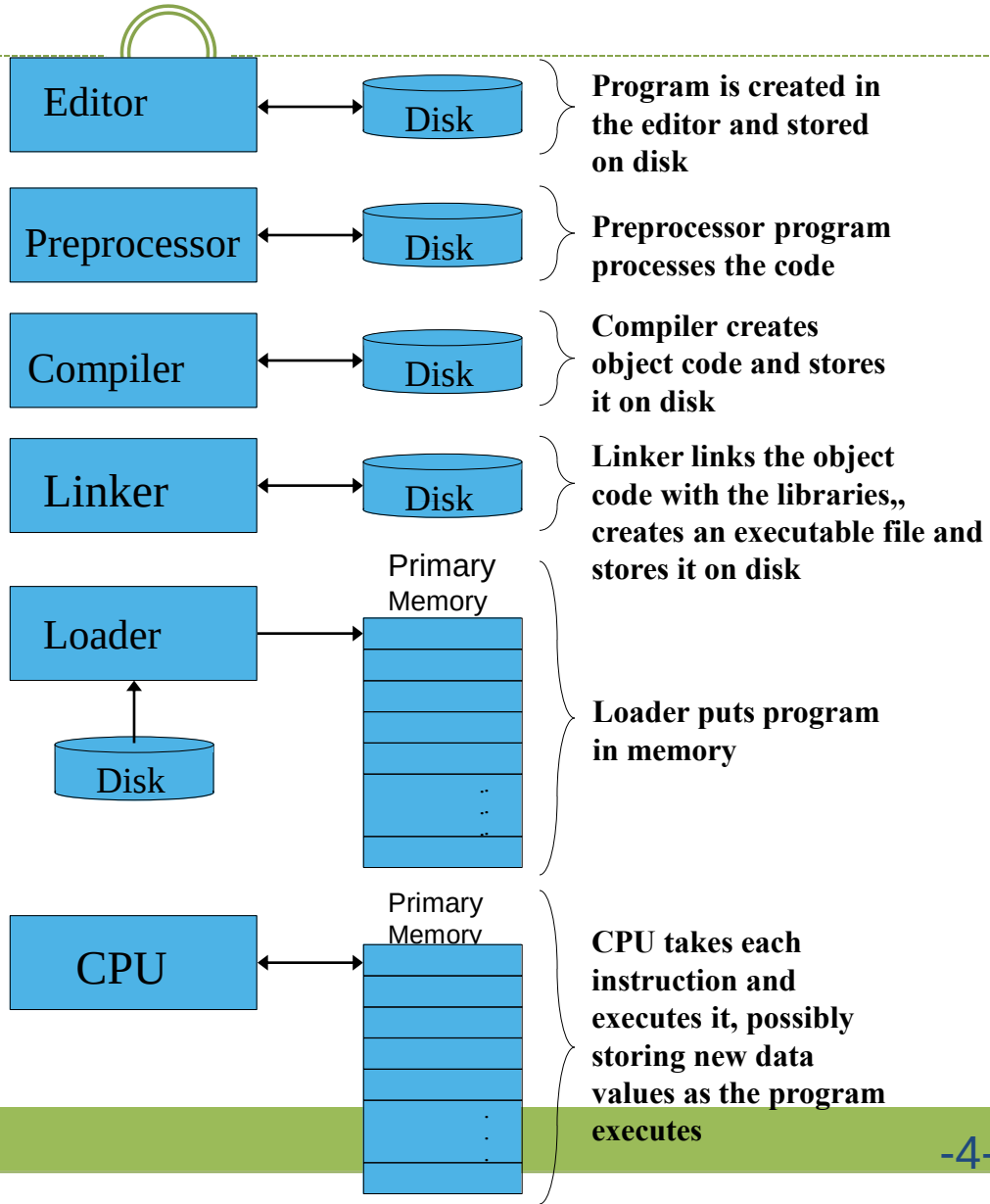
Creating, Compiling, and Running Programs

- A program written in a high-level language is called a ***source program***
- Program is **pre-processed**
- Since a computer cannot understand a source program Program called a ***compiler*** is used to translate the source program into a machine language program called an ***object program (.obj)***
- The object program is often then **linked** with other supporting **library code** to generate an **executable file (.exe)**

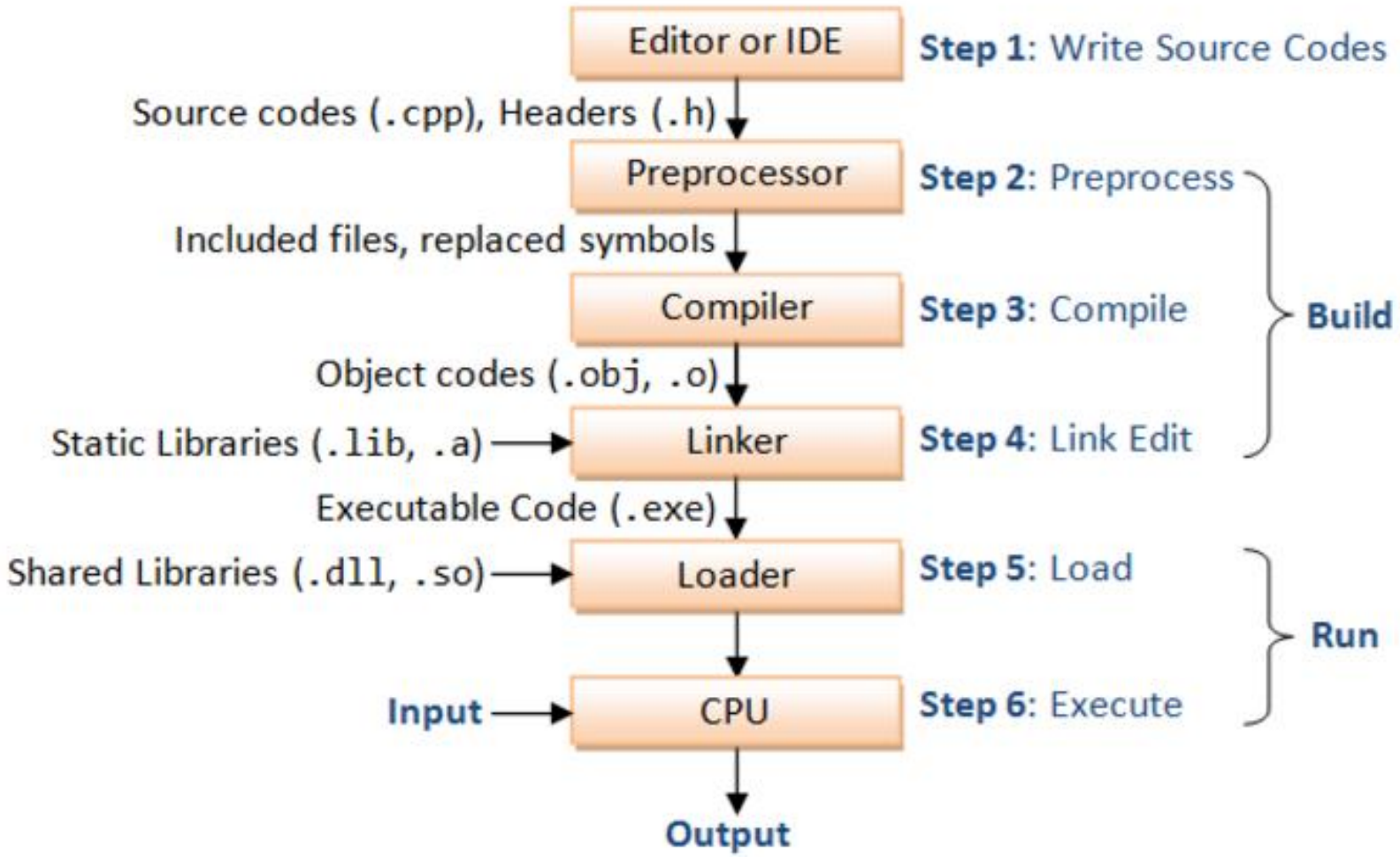
Basics of a Typical C++ Environment

C++ Program Phases:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Basics of a Typical C++ Environment



A Simple C++ Program

A simple C++ program that displays the message *Welcome to C++* on the console.

```
#include <iostream>
int main()
{
  std::cout << "Welcome to C++!" ;
  return 0;
}
```

Program Execution

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout<<"Welcome to C++!";
6      return 0;
7  }
```

C:\Windows\system32\cmd.exe

```
Welcome to C++!Press any key to continue . . .
```

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout<<"Welcome to C++!"<<std::endl;
6      return 0;
7  }
```

C:\Windows\system32\cmd.exe

```
Welcome to C++!
Press any key to continue . . .
```

```
1   // Fig. 1.2:
2   // A first pr
3   #include <iostream>
4
5   // function main b
6   int main()
7   {
8      std::cout << "Welcome to C++"
9
10     return 0;    // indicate that
11
12  }  // end function main
```

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

Function `main` returns an integer value.

Single-line comments.

Preprocessor directive to include input/output stream header file `<iostream>`.

Left brace `{` begins function body.

Function `main` appears exactly once in every C++ program.

Statements end with a semicolon `;`

Stream insertion operator.

Name `cout` belongs to namespace `std`.

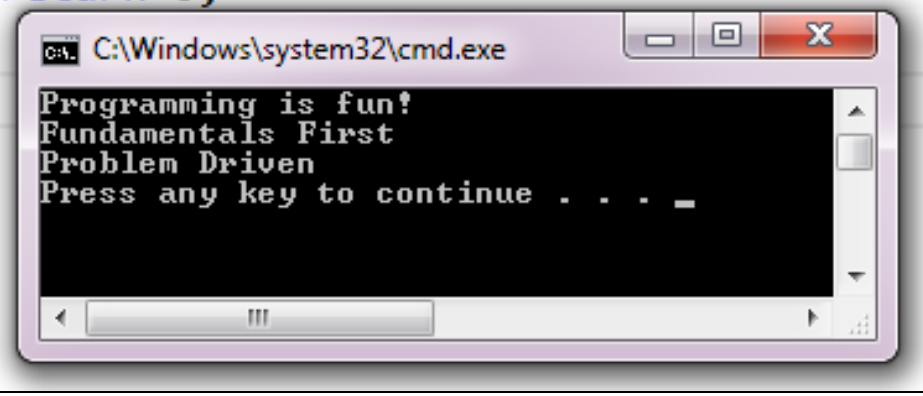
Keyword `return` is one of several means to exit function; value `0` indicates program terminated successfully.

Corresponding right brace `}` ends function body.

Extending the Simple C++ Program

You can rewrite the program to display three messages using endl.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5     cout << "Programming is fun!" << endl;
6     cout << "Fundamentals First" << endl;
7     cout << "Problem Driven" << endl;
8     return 0;
9  }
10
```



Computing with Numbers

Further, you can perform mathematical computations and displays the result to the console.

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      cout<< "(10.5 + 2 * 3) / (45 - 3.5) = ";
6      cout<< (10.5 + 2 * 3) / (45 - 3.5) << endl;
7      return 0;
8  }
9
```

C:\Windows\system32\cmd.exe

```
(10.5 + 2 * 3) / (45 - 3.5) = 0.39759
Press any key to continue . . .
```

Identifiers

- *Valid identifier* is a sequence of one or more letters, digits or underscore characters _.
- Spaces, punctuation marks and symbols cannot be part of an identifier.
- Variable identifiers always have to begin with a letter or _.
- When inventing your own identifiers, they cannot match any keyword of the C++ language.

Some C/C++ Keywords

FYI

alignas	decltype	namespace	struct
alignof	default	new	switch
and	delete	noexcept	template
and_eq	double	not	this
asm	do	not_eq	thread_local
auto	dynamic_cast	nullptr	throw
bitand	else	operator	true
bitor	enum	or	try
bool	explicit	or_eq	typedef
break	export	private	typeid
case	extern	protected	typename
catch	false	public	union
char	float	register	unsigned
char16_t	for	reinterpret_cast	using
char32_t	friend	return	virtual
class	goto	short	void
compl	if	signed	volatile
const	inline	sizeof	wchar_t
constexpr	int	static	while
const_cast	long	static_assert	xor
continue	mutable	static_cast	xor_eq

Identifiers

- It is important to choose a name that is *self-descriptive* and closely reflects the meaning of the variable, e.g., numberOfStudents or numStudents.
- Do not use meaningless names like a, b, c, d, i, j, k, i1, j99.
- Avoid single-alphabet names, which is easier to type but often meaningless, unless they are common names like x, y for coordinates, i for index.
- It is perfectly okay to use long names of say 30 characters to make sure that the name accurately reflects its meaning!
- Use singular and plural nouns prudently to differentiate between singular and plural variables. For example, you may use the variable row to refer to a single row number and the variable rows to refer to many rows.

Identifiers

- The C++ language is a *"case sensitive"* language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters.
- For example:

  RESULT variable is not the same as

  result variable or

  Result variable

  These are three different variable identifiers.

Notes

- You need to declare the name of a variable before it can be used.
- C++ is a "strongly-type" language. A variable takes on a type. Once the *type* of a variable is declared, it can only store a value belonging to this particular type. For example, an int variable can hold only integer such as 123, and NOT floating-point number such as -2.17 or text string such as "Hello".
- Each variable can only be declared once.
- In C++, you can declare a variable anywhere inside the program, as long as it is declared before used.
- The type of a variable cannot be changed inside the program.

Exercise

Identifier	Valid?
sumX2	valid
Hourly_rate	valid
Gross Pay	invalid
“X”	invalid
name@	invalid
_myVariable	valid
@address	invalid

Identifier	Valid?
Net-salary	invalid
4grade	invalid
numStudents	valid
while	invalid
Employee’s	invalid
20morrow	invalid
Final_grade_90	valid

Memory Concepts

- Variables
  - Correspond to actual locations in computer's memory
  - Every variable has name, type, size and value
  - When new value is placed/assigned to variable, it overwrites previous value
  - Reading variables from memory nondestructive

x = 5

letter = ‘d’

sal = 2560.50

Assignment Operator

- The assignment operator (=) is a binary operator (two operands).
- It deals with a variable on the left side and a value on the right side.

```
x =  5;
m = m +30;
k =  j*f + s;
d  =  (a-c) * (v/x);
```

Memory Concepts

```
std::cin >> integer1;
```

- Assume user entered 45

integer1	45

```
std::cin >> integer2;
```

- Assume user entered 72

integer1	45
integer2	72

```
sum = integer1 + integer2;
```

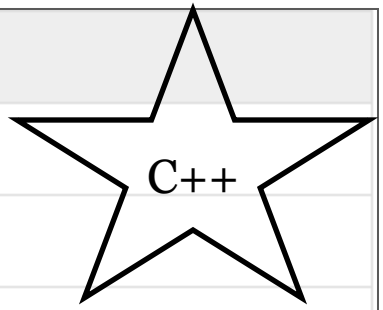
integer1	45
integer2	72
sum	117

Basic C++ Data Types

- You need to use various variables to store various information.
- Variables are reserved memory locations to store values.
- Based on the data type of a variable, the operating system allocates memory.
- A byte is the minimum amount of memory that we can manage in C++. A byte can store a relatively small amount of data: one single character or a small integer (generally an integer between 0 and 255).

Basic C++ Data Types

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double



- As the name implies, a double has 2x the precision of float. A double has 15 decimal digits of precision, while float has 7.

Variable Declaration

- In order to use a variable in C++, we must first declare it specifying its data type.
- To declare a new variable: Write the specifier of the desired data type (like int, bool, float…) followed by a valid variable identifier. Examples:
  - float salary;
  - char letter;
- To declare more than one variable of the same type, declare all of them in a single statement by separating their identifiers with commas.
  - int a, b, c;

Basic C++ Data types

FYI

Type	Typical Bit Width	Typical Range
char	1byte	-128 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-128 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-2,147,483,648 to 2,147,483,647
signed long int	4bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)

Arithmetic

- Arithmetic calculations
  - `*`
    - Multiplication
  - `/`
    - Division
    - Integer division truncates remainder
      - `7 / 5` evaluates to 1
  - `%`
    - Modulus operator returns remainder
      - `7 % 5` evaluates to 2

Arithmetic

- Rules of operator precedence
  - Operators in parentheses evaluated first
    - Nested/embedded parentheses
      - Operators in innermost pair first
  - Multiplication, division, modulus applied next
    - Operators applied from left to right
  - Addition, subtraction applied last
    - Operators applied from left to right

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

```
1   // Fig. 1.6: fig01_06.cpp
2   // Addition program.
3   #include <iostream>
4   using namespace std;
5   // function main begins program execution
6   int main()
7   {
8      int integer1;  // first number to be input by user
9      int integer2;  // second n
10     int sum;       // variable
11
12     cout << "Enter first integ
13     cin >> integer1;
14
15     cout << "Enter second integer\n"; // prompt
16     cin >> integer2;
17
18     sum = integer1 + intege
19
20     cout << "Sum is " << sum << endl; // print sum
21
22     return 0;   // indicate that program ended successfully
23
24  } // end function main
```

Declare integer variables.

Use stream extraction operator with standard input stream to obtain user input.

Calculations can be performed in output statements: alternative for lines 18 and 20:
`std::cout << "Sum is "<<integer1 + integer2 << std::endl;`

Stream manipulator `std::endl` outputs a newline

Concatenating, chaining or cascading stream insertion operations.

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Escape Characters

- \
- Indicates “special” character output

Escape Sequence	Value
\n	newline
\t	horizontal tab
\b	backspace
\r	carriage return
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

Escape Characters

```
int main() {
    char newline = '\n';
    char tab = '\t';
    char backspace = '\b';
    char backslash = '\\';
    char nullChar = '\0';

    cout << "Newline character: " << newline << "ending" << endl; // Newline character:
                                                                  //  ending
    cout << "Tab character: " << tab << "ending" << endl; // Tab character : ending
    cout << "Backspace character: " << backspace << "ending" << endl; // Backspace character : ending
    cout << "Backslash character: " << backslash << "ending" << endl; // Backslash character : \endin
    cout << "Null character: " << nullChar << "ending" << endl; //Null character:  ending
}
```

C:\Windows\system32\cmd.exe

```
Newline character:
ending
Tab character:  ending
Backspace character:ending
Backslash character: \ending
Null character:  ending
Press any key to continue . . .
```

```cpp
// Fig. 1.4: fig01_04.cpp
// Printing a line with multiple statements.
#include <iostream>
using namespace std;
// function main begins program execution
int main()
{
   cout << "Welcome ";
   cout << "to C++ \n";

   return 0;   // indicate that program ended successfully

} // end function main
```

Multiple stream insertion statements produce one line of output.

```
Welcome to C++
```

```
// Fig. 1.5: fig01_05.cpp
// Printing multiple lines with a single statement
#include <iostream>
using namespace std;
// function main begins program executio
int main()
{
   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++
```

Compound Assignment Operators

- **Compound assignment (+=, -=, *=, /=)**

  When we want to modify the value of a variable by performing an operation on the value currently stored in that variable; we can use compound assignment operators:

Expression	Equivalent to
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

Decision Making: Equality and Relational Operators

Standard algebraic operator	C++ operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
$>$	`>`	`x > y`	is **x** greater than **y** ?
$<$	`<`	`x < y`	is **x** less than **y** ?
$\geq$	`>=`	`x >= y`	is **x** greater than or equal to **y** ?
$\leq$	`<=`	`x <= y`	is **x** less than or equal to **y** ?
Equality operators			
$=$	`==`	`x == y`	is **x** equal to **y** ?
$\neq$ < >	`!=`	`x != y`	is **x** not equal to **y**?

Decision Making

```
if (condition)
        statement;

if (condition)
{
        statement1;
        statement2;
}

if (condition)
        statement1;
else
        statement2;
```

```
if (condition)
{
        statement1;
        statement2;
}
else
{
        statement3;
        statement4;
}
```

```
// Fig. 1.14: fig01_14.cpp
// Using if statements, relational
// operators, and equality operators.
#include <iostream>

using namespace std;


// function main begins program execution
int main()
{
   int num1;  // first n
   int num2;  // second

   cout << "Enter two integers, and I will tell you\n"
        << "the relationships they satisfy: ";
   cin >> num1 >> num2;   // read two integers

   if ( num1 == num2 )
      cout << num1 << " is equal to " << num2 << endl;

   if ( num1 != num2 )
      cout << num1 << " is not equal to " << num2 << endl;

```

`using` statement eliminates need for `std::` prefix.

Declare variables.

Can write `cout` and `cin` without `std::` prefix.

`if` structure compares values of `num1` and `num2` to test for equality.

If condition is true (i.e., values are equal), execute this statement.

`if` structure compares values of `num1` and `num2` to test for inequality.

If condition is true (i.e., values are not equal), execute this statement.

```
26     if ( num1 < num2 )
27        cout << num1 << " is less than " << num2 << endl;
28
29     if ( num1 > num2 )
30        cout << num1 << " is greater than " << num2 << endl;
31
32     if ( num1 <= num2 )
33        cout << num1 << " is less than or equal to "
34             << num2 << endl;
35
36     if ( num1 >= num2 )
37        cout << num1 << " is greater than or equal to "
38             << num2 << endl;
39
40     return 0;   // indicate that program ended successfully
41
42  } // end function main
```

Statement may be split over several lines.

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

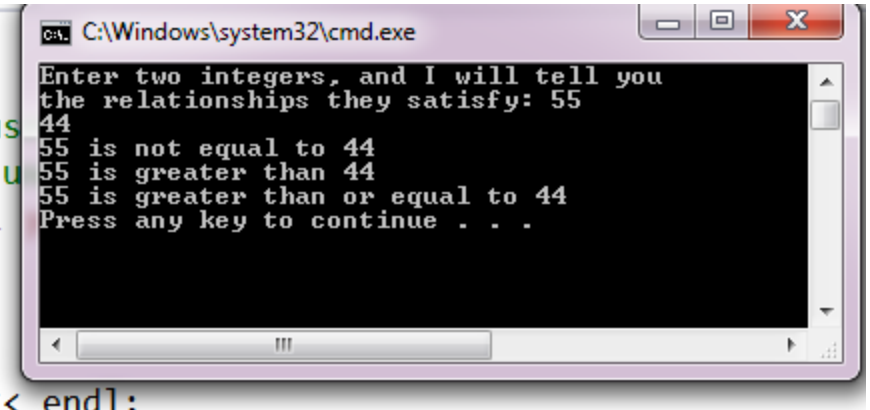
This program contains separate IF conditions, nothing stops the execution of multiple IF statements for the input numbers.

Different from IF-ELSE IF-ELSE construct.

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Program Execution

```
int main()
{
   int num1;  // first number to be read from us
   int num2;  // second number to be read from u
   cout << "Enter two integers, and I will tell
        << "the relationships they satisfy: ";
   cin >> num1 >> num2;   // read two integers
   if ( num1 == num2 )
      cout << num1 << " is equal to " << num2 << endl;
   if ( num1 != num2 )
      cout << num1 << " is not equal to " << num2 << endl;
   if ( num1 < num2 )
      cout << num1 << " is less than " << num2 << endl;
   if ( num1 > num2 )
      cout << num1 << " is greater than " << num2 << endl;
   if ( num1 <= num2 )
      cout << num1 << " is less than or equal to " << num2 << endl;
   if ( num1 >= num2 )
      cout << num1 << " is greater than or equal to " << num2 << endl;
   return 0;   // indicate that program ended successfully
} // end function main
```



2 players … 1 winner

```
#include <iostream>
using namespace std;
int main()
{
 int score1, score2, winner;
 cout << "Enter scores of the two players:\n";
 cin >> score1 >> score2;

 if (score1 > score2)
    winner = 1;
 else
    winner = 2;
 cout << "Winner is player number: " << winner;
 return 0;
}
```

2 players … 1 winner

```
#include <iostream>
using namespace std;
int main()
{
 int score1, score2, winner;
 cout << "Enter scores of the two players:\n";
 cin >> score1 >> score2;

 if (score1 > score2)
     cout << "Winner is player number 1" << endl;
 else if (score1 < score2)
     cout << "Winner is player number 2" << endl;
 else
    cout << "Tie" << endl;
 return 0;
}
```

Braces

- If we want more than a single statement to be executed in case that the condition is true we can specify a block using braces { }

```
if (x == 100)
{
    cout << "x is ";
    cout << x;
}
```

2 players … 1 winner

```
int score1, score2, winner;
cout << "Enter scores of the two players:\n";
cin >> score1 >> score2;
if (score1 > score2)
{
    cout << "Winner is player number";
     cout << "1" << endl;
}
else if (score1 < score2)
{
    cout << "Winner is player number";
     cout << "2" << endl;
}
else
   cout << "Tie" << endl;
```

Errors

- Syntax Errors
- Logic Errors
- Runtime Errors

Syntax Error

Detected at compile time and you won't be able to run your program until these are fixed.

- Not following the grammatical rules used in declaration of identifier.
- Not declaring an identifier used in program.
- Not terminating statement by semicolon.
- Not providing equal number of opening and closing braces etc.

These errors can be corrected by the user as it is displayed while compiling the program.

Common Syntax Errors

- Missing Braces
- Missing Semicolons
- Missing Quotation Marks
- Misspelling Names

Exercise

```
#include <iostream>
using namespace std

int main()
{
  cout << "Programming is fun << endl;

  return 0;
}
```

Terminate Using statement with ;

Close the string properly with "

Syntax Error

FirstProgram.cpp

(Global Scope)

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, world" << endl;

    x = 5;

    system("pause");

    return 0;
```

100 %

Output

Show output from: Build

```
1>------ Build started: Project: FirstProgram, Configuration: Debug Win32 ------
1>  FirstProgram.cpp
1>c:\users\greg\documents\visual studio 2010\projects\firstprogram\firstprogram\firstprogram.cpp(8): error C2065: 'x' : undeclared identifier
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

Logic Error

- This error won’t be displayed on the screen.
- It is caused by incorrect business logic.
- It will lead to display wrong results. Example: An infinite loop.

Exercise

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Celsius 35 is Fahrenheit degree " << endl;
  cout << (9 / 5) * 35 + 32 << endl;

  return 0;
}
```

Answer: 67
Problem: Integer division truncates remainder
Correct answer: 95

Runtime Error

This error occurs while running a program caused by

- Division by 0
- Overflow:

  Exceed the maximum value a data type can have. E.g.: When a variable is assigned a value that is too large to be stored.
- Underflow:

  Larger negative exponents not being available to represent the number.

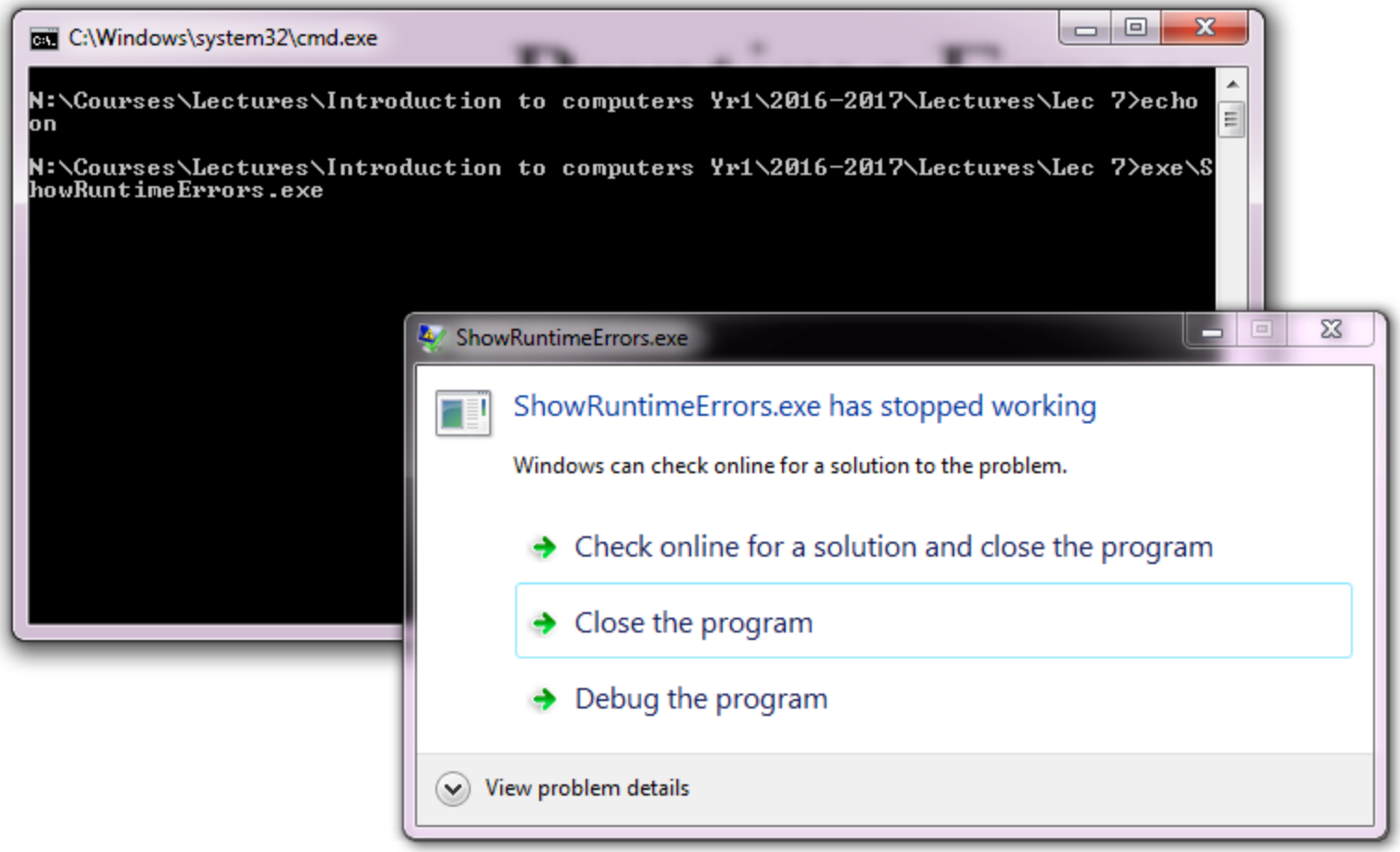
Exercise

```
#include <iostream>
using namespace std;

int main()
{
  int i = 4;
  int j = 0;
  cout << i / j << endl;

  return 0;
}
```

Runtime Error: Division by Zero



Exercise: Correct Code

```
{
    int a, b;
    cin<<a;
    b=a;
    cout <<"b=", a;
};
```

```
{
    int a, b;
    cin>>a;
    b=a;
    cout <<"b="<<a;
}
```

Exercise: Correct Code

```
{
  cin >> x;
  cout << x;
  //...
}
```

```
{
  int x;
  cin >> x;
  cout << x;
  //...
}
```

'x' : undeclared identifier

The compiler doesn't know what x means. You need to declare it as a variable.

Exercise: Correct Code

```
{
    int x;
    cin >> x;
    if (x == 5)
    {
        cout << x;
        cout<< x*2;
}
```

fatal error C1075: end of file found before the left brace '{'
…

```
{
    int x;
    cin >> x;
    if (x == 5)
    {
        cout << x;
        cout<< x*2;
    }
}
```

When structure consists of many statements, enclose them with { }

Exercise: Correct Code

```
int a, b;
int sum = a + b;
cout << "Enter two numbers ";
cin >> a;
cin >> b;
cout << "The sum is: " << sum;
```

```
Run:
Enter two numbers to add: 1
3
The sum is: -1393
```

warning C4700: uninitialized local variable ‘a' used
warning C4700: uninitialized local variable ‘b' used

```
int a, b;
int sum;
cout << "Enter two numbers ";
cin >> a;
cin >> b;
sum = a + b;
cout << "The sum is: "<< sum;
```

To fix this error, move the addition step after the input line.

Exercise: Correct Code

```
int count;
if (count < 100)
{
   cout << count;
   count=3250;
}
```

warning C4700: uninitialized local variable 'count' used

Runtime:

The variable ‘count’ is being used without being initialized

```
int count = 0;
while (count < 100)
{
   cout << count;
   count=3250;
}
```

Exercise: True/False

1. Writing variables to memory is nondestructive. ✗
2. If we want more than a single statement to be executed in case that the condition is true we can specify a block using braces { } ✓
3. Division by zero and overflow are logic errors. ✗
4. To declare more than one variable of the same type, declare all of them in a single statement by separating their identifiers with commas. ✓

Exercise: Complete

1. Based on the ---------- of a variable, the operating system allocates memory.
2. Declarative region that provides a scope to identifiers is called ----------.
3. A ---------- is the minimum amount of memory that we can manage in C++.
4. ---------- creates object code and stores it on disk.
5. The object program is often then linked with other supporting library code to generate ----------
6. ---------- indicates a single-line comment.

Problem: Computing Loan Payments

This program lets the user enter the yearly interest rate, number of years, and loan amount and computes monthly payment and total payment. Monthly payment can be computed using:

$$\frac{loanAmount * monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numYears*12}}}$$

Solution: Computing Loan Payments

```
float annualInterestRate, monthlyInterestRate, loanAmount;
int numberOfYears;

cout << "Enter annual interest rate percent, for example 8.25: ";
cin >> annualInterestRate;
monthlyInterestRate = annualInterestRate / 1200;

cout << "Enter number of years as an integer, for example 5: ";
cin >> numberOfYears;
cout << "Enter loan amount, for example 120000.95: ";
cin >> loanAmount;

float monthlyPayment = loanAmount * monthlyInterestRate /
(1 - 1 / pow (1 + monthlyInterestRate, numberOfYears * 12));
float totalPayment = monthlyPayment * 12 * numberOfYears;

cout << "The monthly payment is " << monthlyPayment << endl <<
  "The total payment is " << totalPayment << endl;
```

`pow(x,y)` = x raised to the yth power.

Thank You