The Fundamentals of C++

Basic programming elements and concepts

Program Organization

- Program statement
  - Definition
  - Declaration
  - Action
- Executable unit
  - Named set of program statements
  - Different languages refer to executable units by different names
    - Subroutine: Fortran and Basic
    - Procedure: Pascal
    - Function : C++
    - Method : Java

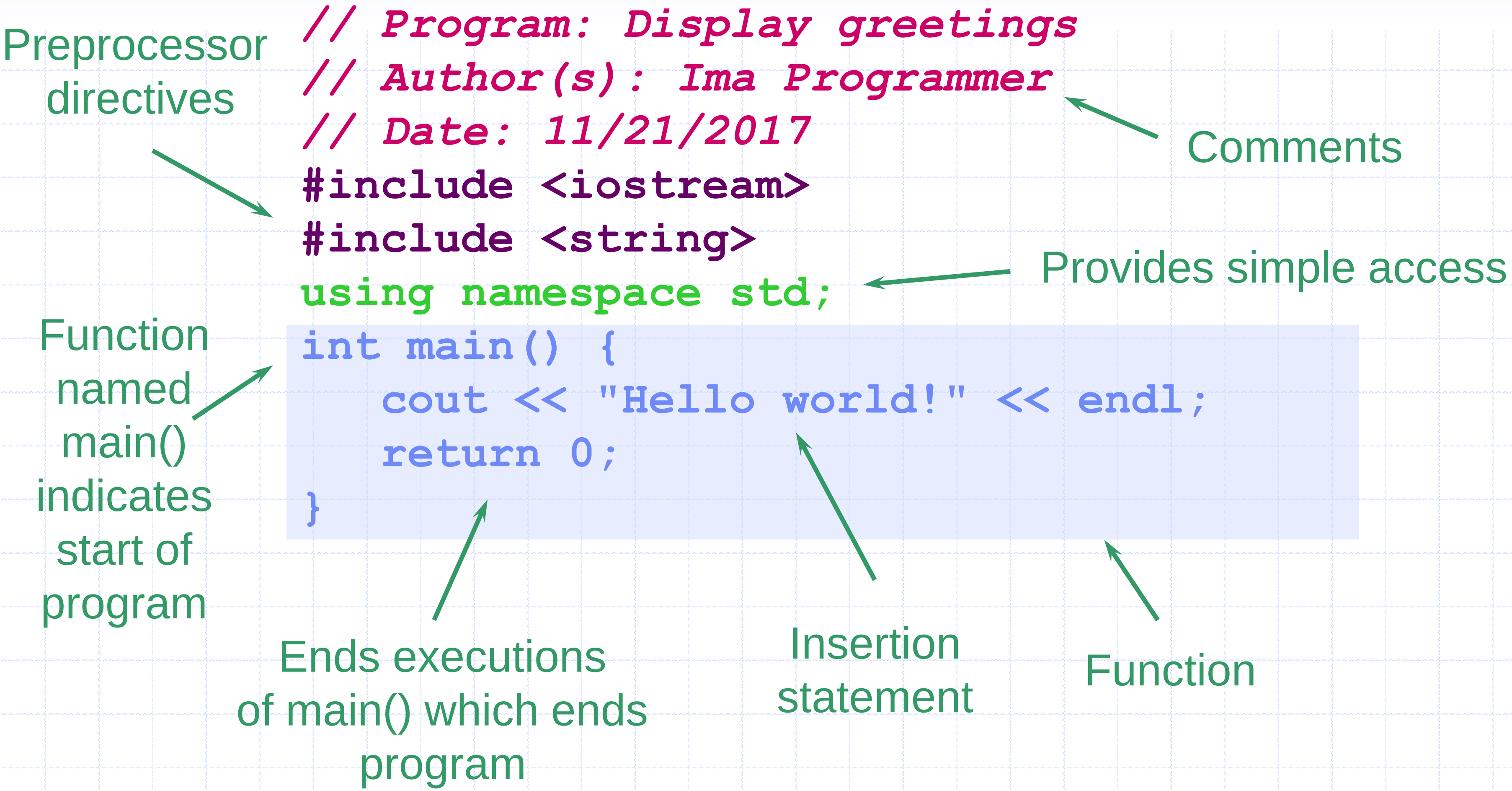
Program Organization

- C++ program
  - Collection of definitions, declarations and functions
  - Collection can span multiple files
- Advantages
  - Structured into small understandable units
  - Complexity is reduced
  - Overall program size decreases

Object

- Object is a representation of some information
  - Name
  - Values or properties
    - Data members
  - Ability to react to requests (messages)!!
    - Member functions
- When an object receives a message, one of two actions are performed
  - Object is directed to perform an action
  - Object changes one of its properties

A First Program - Greeting.cpp



```
// Program: Display greetings
// Author(s): Ima Programmer
// Date: 11/21/2017
#include <iostream>
#include <string>
using namespace std;
int main() {
   cout << "Hello world!" << endl;
   return 0;
}
```

Preprocessor directives

Comments

Provides simple access

Function named main() indicates start of program

Ends executions of main() which ends program

Insertion statement

Function

Processing a C++ Program

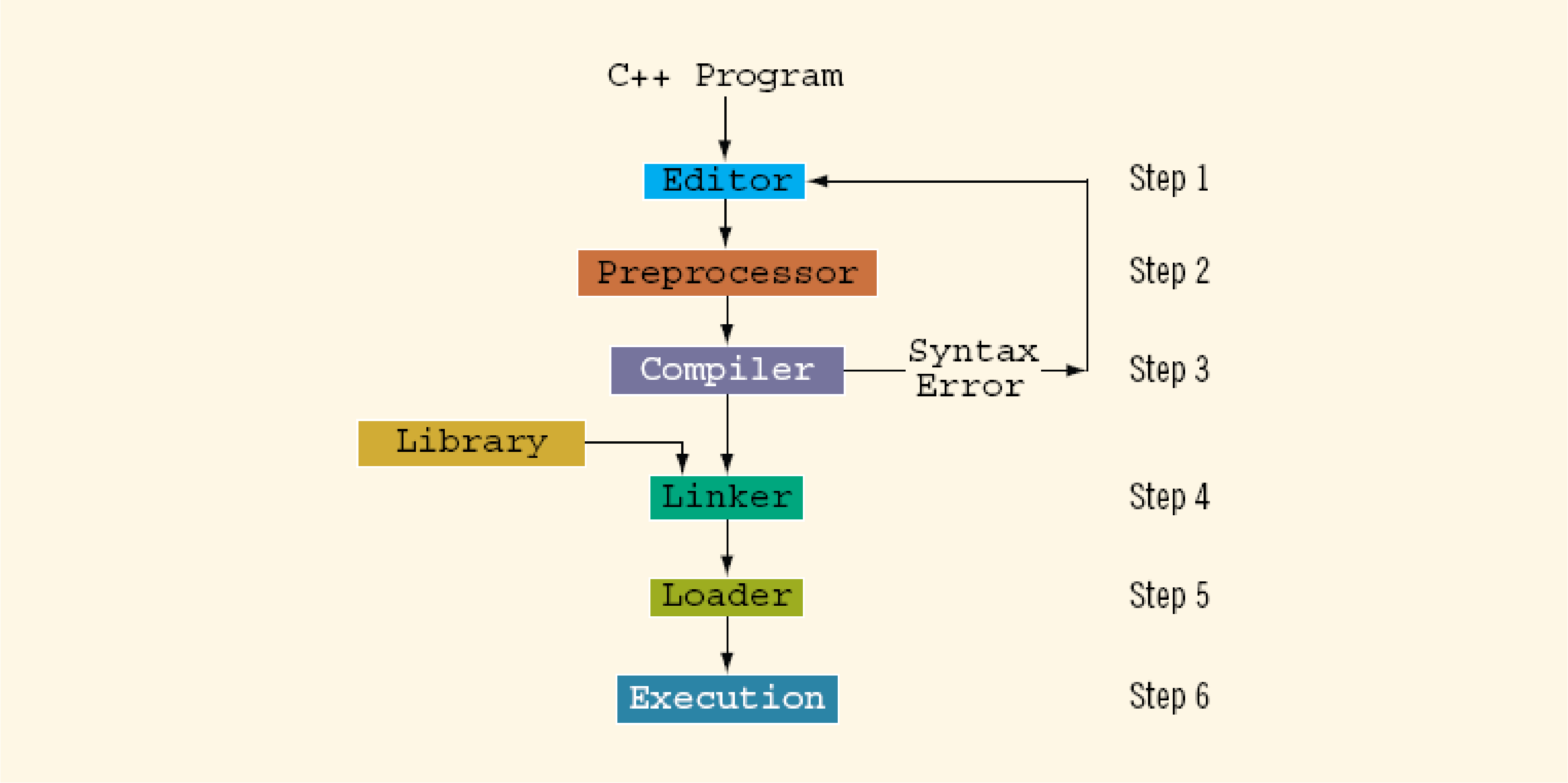
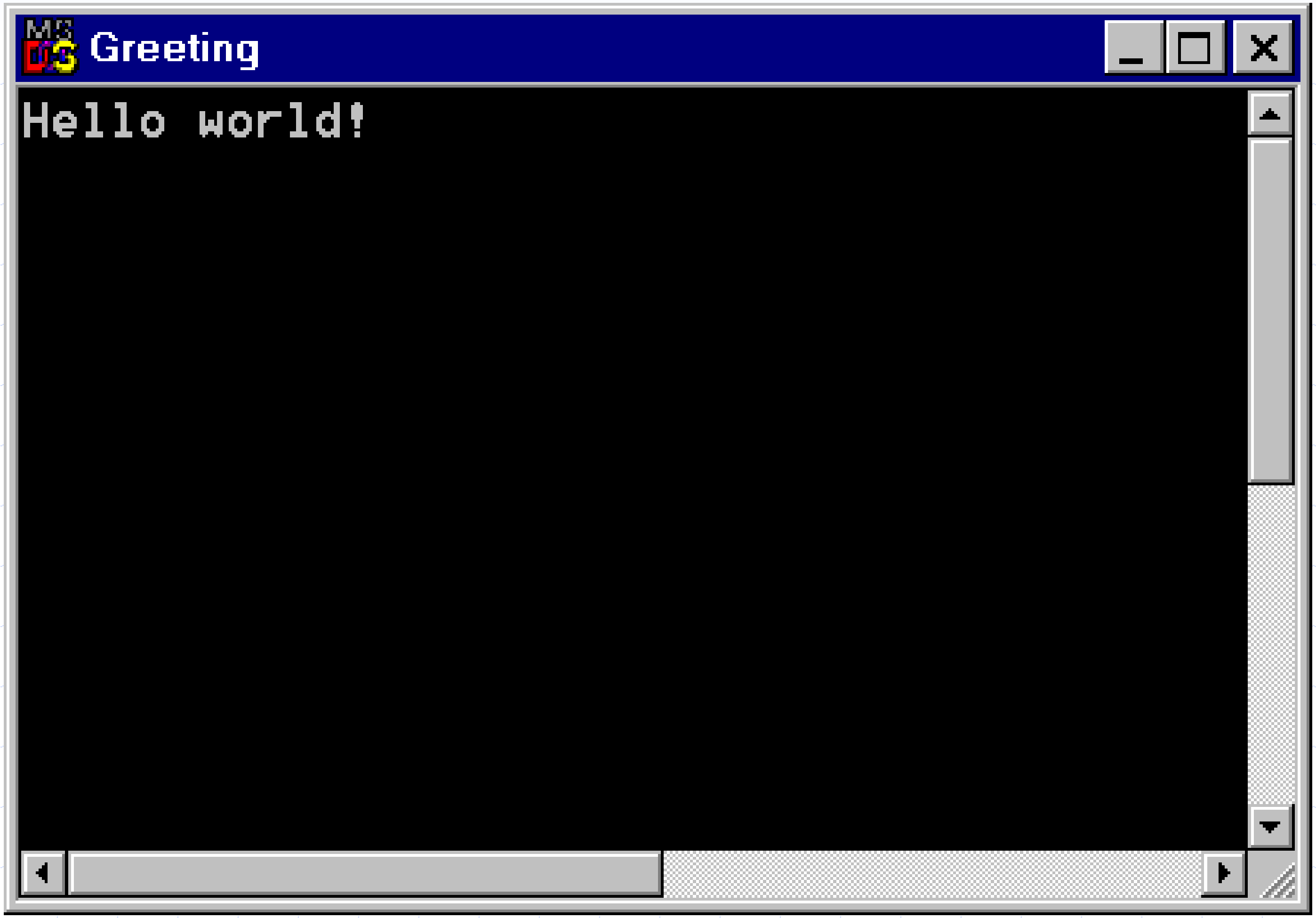


FIGURE 1-3 Processing a C++ program

Greeting Output



```
Hello world!
```

Area.cpp

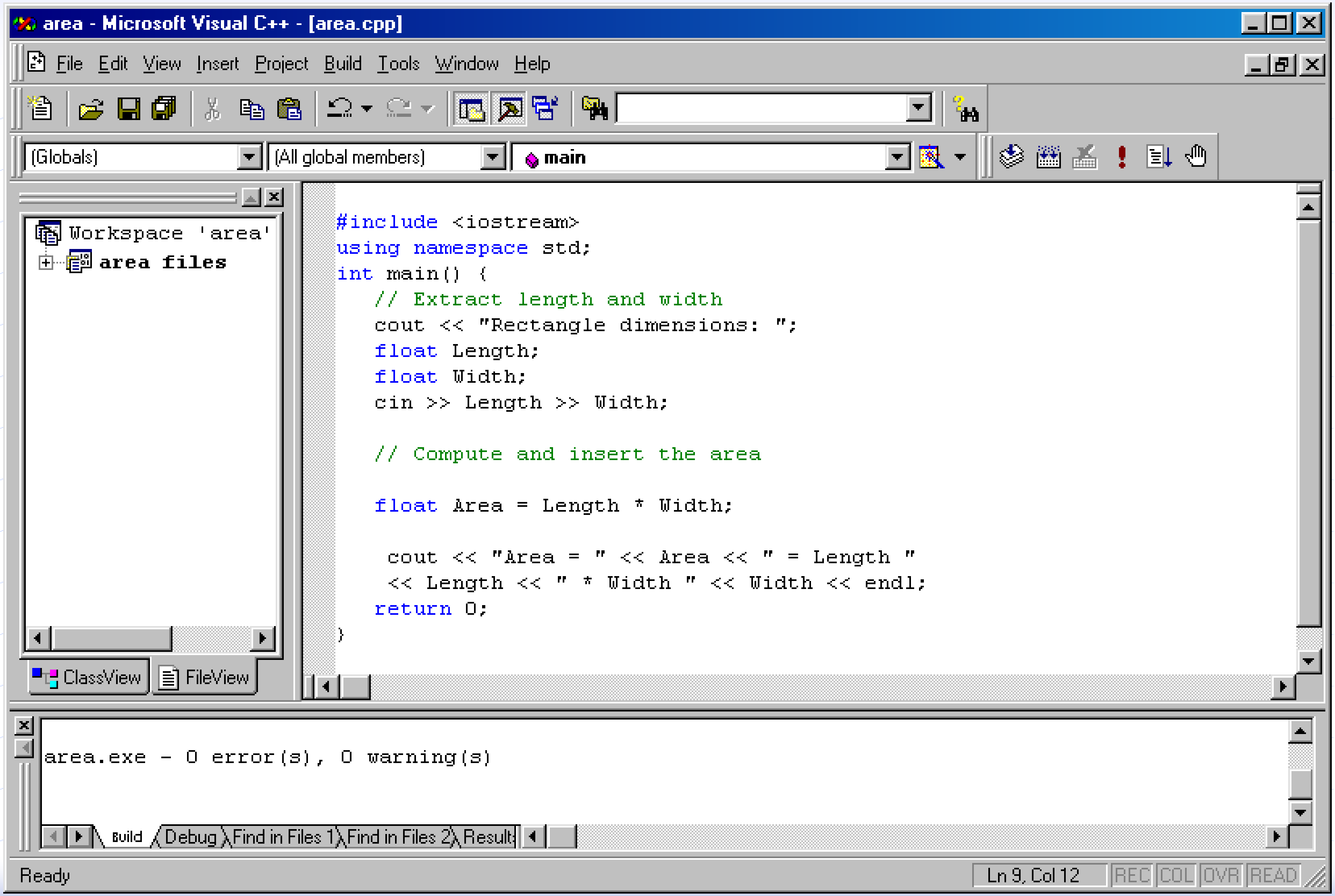
```
#include <iostream>
using namespace std;
int main() {
   // Extract length and width
   cout << "Rectangle dimensions: ";
   float Length;
   float Width;
   cin >> Length >> Width;

   // Compute and insert the area

   float Area = Length * Width;

  cout << "Area = " << Area << " = Length "
    << Length << " * Width " << Width << endl;
   return 0;
}
```

Definitions

Extraction

Definition with initialization

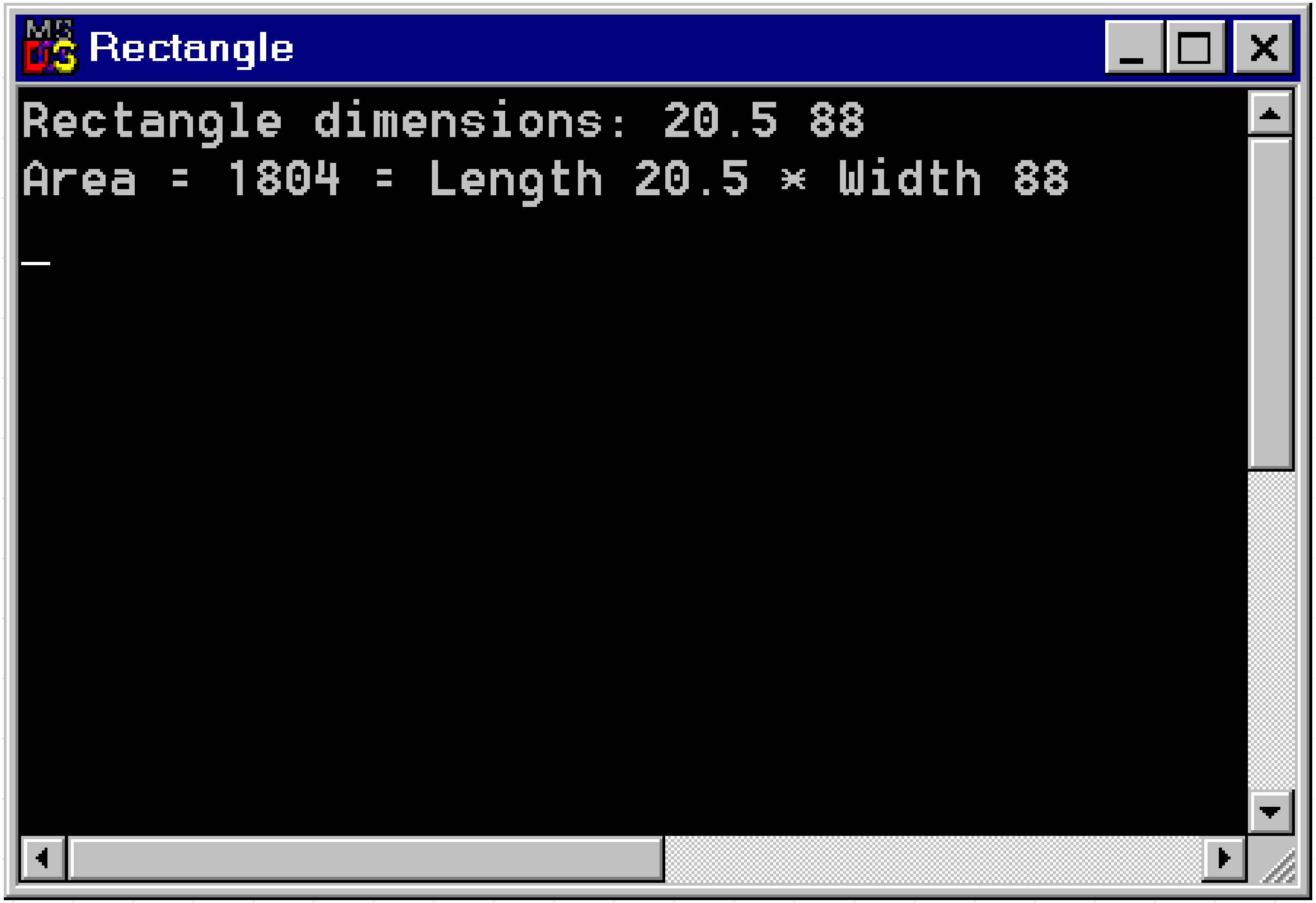
Visual C++ IDE with Area.cpp



```
#include <iostream>
using namespace std;
int main() {
   // Extract length and width
   cout << "Rectangle dimensions: ";
   float Length;
   float Width;
   cin >> Length >> Width;

   // Compute and insert the area

   float Area = Length * Width;

    cout << "Area = " << Area << " = Length "
    << Length << " * Width " << Width << endl;
   return 0;
}
```

```
area.exe - 0 error(s), 0 warning(s)
```

Area.cpp Output



```
Rectangle dimensions: 20.5 88
Area = 1804 = Length 20.5 × Width 88
```

Comments

- Allow prose or commentary to be included in program
- Importance
  - Programs are read far more often than they are written
  - Programs need to be understood so that they can be maintained
- C++ has two conventions for comments
  - `//` single line comment (preferred)
  - `/*` long comment `*/` (save for debugging)
- Typical uses
  - Identify program and who wrote it
  - Record when program was written
  - Add descriptions of modifications

Fundamental C++ Objects

- C++ has a large number of fundamental or built-in object types
- The fundamental object types fall into one of three categories
  - Integer objects
  - Floating-point objects
  - Character objects

Z

5 1.28345

1 3.14

P

Integer Object Types

- The basic integer object type is `int`
  - The size of an `int` depends on the machine and the compiler
    - On PCs it is normally 16 or 32 bits
- Other integers object types
  - `short`: typically uses less bits
  - `long`: typically uses more bits
- Different types allow programmers to use resources more efficiently
- Standard arithmetic and relational operations are available for these types

Integer Constants

- Integer constants are positive or negative whole numbers
- Integer constant forms
  - Decimal
  - Octal (base 8)
    - Digits 0, 1, 2, 3, 4, 5, 6, 7
  - Hexadecimal (base 16)
    - Digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a , b, c, d, e, f, A, B, C, D, E, F

Decimal Constants

- Examples
  - 97
  - 40000L
  - 50000
  - 23a (illegal)

L or l indicates long integer

- The type of the constant depends on its size, unless the type specifier is used

Character Object Types

- Character type `char` is related to the integer types
- Characters are encoded using a scheme where an integer represents a particular character
- ASCII is the dominant encoding scheme
  - Examples
    - `' '` encoded as 32
    - `'A'` encoded as 65
    - `'a'` encoded as 97
    - `'+'` encoded as 43
    - `'Z'` encoded as 90
    - `'z'` encoded as 122

Character Operations

- Arithmetic and relational operations are defined for characters types
  - `'a' < 'b'` is true
  - `'4' > '3'` is true
  - `'6' <= '2'` is false

Character Constants

- Explicit (literal) characters within single quotes
  - `'a'`,`'D'`,`'*'`
- Special characters - delineated by a backslash \
  - Two character sequences (escape codes)
  - Some important special escape codes
    - `\t` denotes a tab
    - `\\` denotes a backslash
    - `\"` denotes a double quote
    - `\n` denotes a new line
    - `\'` denotes a single quote
  - `'\t'` is the explicit tab character, `'\n'` is the explicit new line character, and so on

Literal String Constants

- A literal string constant is a sequence of zero or more characters enclosed in double quotes
  - `"We are even loonier than you think"`
  - `"Rust never sleeps\n"`
  - `"Nilla is a Labrador Retriever"`

- Not a fundamental type

Floating-Point Object Types

- Floating-point object types represent real numbers
  - Integer part
  - Fractional part
- The number 108.1517 breaks down into the following parts
  - 108 - integer part
  - 1517 - fractional part
- C++ provides three floating-point object types
  - `float`
  - `double`
  - `long double`

Floating-Point Constants

- Standard decimal notation

  134.123

  0.15F

F or f indicates single precision floating point value

- Standard scientific notation

  1.45E6

  0.979e-3L

L or l indicates long double floating point value

- When not specified, floating-point constants are of type **`double`**

Names

- Used to denote program values or components
- A valid name is a sequence of
  - Letters (upper and lowercase)
  - Digits
    - A name cannot start with a digit
  - Underscores
    - A name should not normally start with an underscore
- Names are case sensitive
  - MyObject is a different name than MYOBJECT
- There are two kinds of names
  - Keywords
  - Identifiers

Keywords

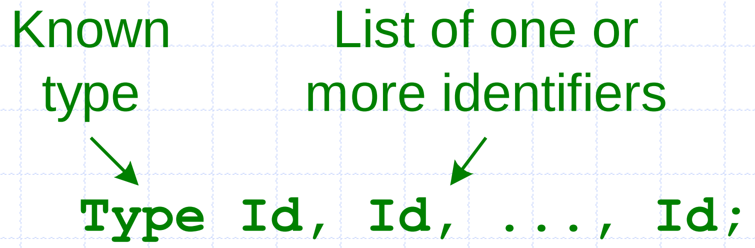
- Keywords are words reserved as part of the language
  - **`int`**, **`return`**, **`float`**, **`double`**
- They cannot be used by the programmer to name things
- They consist of lowercase letters only
- They have special meaning to the compiler

Identifiers

- Identifiers should be
  - Short enough to be reasonable to type (single word is norm)
    - Standard abbreviations are fine (but only standard abbreviations)
  - Long enough to be understandable
    - When using multiple word identifiers capitalize the first letter of each word
- Examples
  - `Min`
  - `Temperature`
  - `CameraAngle`
  - `CurrentNbrPoints`

Definitions

- All objects that are used in a program must be defined
- An object definition specifies
  - Type
  - Name
- General definition form



Known type

List of one or more identifiers

```
Type Id, Id, ..., Id;
```

  - Our convention is one definition per statement!

Examples

```
char Response;
int MinElement;
float Score;
float Temperature;
int i;
int n;
char c;
float x;
```

Objects are uninitialized with this definition form

(Value of an object is whatever is in its assigned memory location)

Arithmetic Operators

- Common

Addition	+
Subtraction	-
Multiplication	*
Division	/
Mod	%

Write **`m*x + b`**
not **`mx + b`**

- Note
  - No exponentiation operator
  - Single division operator
  - Operators are overloaded to work with more than one type of object

Arithmetic Operators

- Integer division
- Operator precedence & associativity
- Initialization with definition
- …

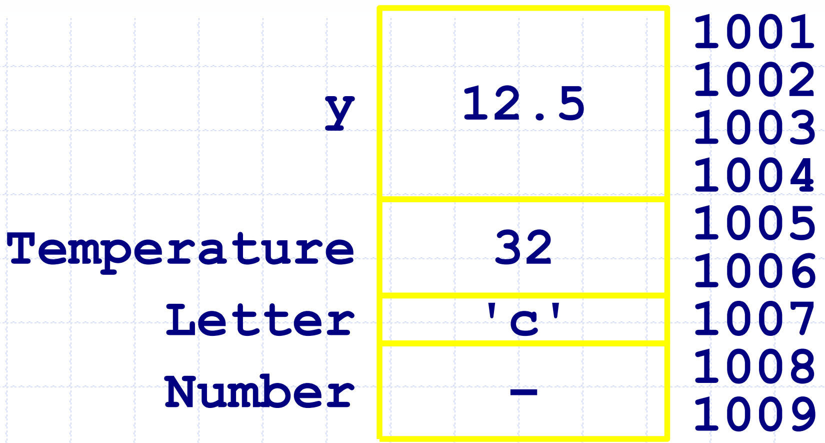
same as in Java

Modifying objects

Operators and Expressions

Memory Depiction

```
float y = 12.5;
int Temperature = 32;
char Letter = 'c';
int Number;
```



y 12.5

Temperature 32

Letter 'c'

Number -

1001
1002
1003
1004
1005
1006
1007
1008
1009

Assignment Statement

Target becomes source

- Basic form
  - *object* = *expression* ;

```
Celsius = (Fahrenheit - 32) * 5 / 9;
y = m * x + b;
```

- Action
  - Expression is evaluated
  - Value of the expression is stored in the object

Definition

```
int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;
```

NewStudents	6
OldStudents	21
TotalStudents	-

Assignment Statement

```
int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;
```

NewStudents	6
OldStudents	21
TotalStudents	27

```
TotalStudents = NewStudents + OldStudents;
```

Assignment Statement

```
int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;
```

NewStudents	6
OldStudents	27
TotalStudents	27

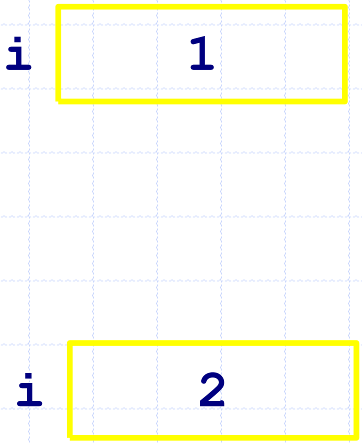
```
TotalStudents = NewStudents + OldStudents;
```

```
OldStudents = TotalStudents;
```

Incrementing

```
int i = 1;
```

```
i = i + 1;
```



Assign the value of expression `i + 1` to `i`

Evaluates to 2

Const Definitions

- Modifier `const` indicates that an object cannot be changed
  - Object is read-only
- Useful when defining objects representing physical and mathematical constants

```
const float Pi = 3.1415;
```

- Value has a name that can be used throughout the program

```
const int SampleSize = 100;
```

- Makes changing the constant easy
  - Only need to change the definition and recompile

Assignment Conversions

- Floating-point expression assigned to an integer object is truncated
- Integer expression assigned to a floating-point object is converted to a floating-point value
- Consider

```
float y = 2.7;
int i = 15;
int j = 10;
i = y;                  // i is now 2
cout << i << endl;
y = j;                  // y is now 10.0
cout << y << endl;
```

Nonfundamental Types

- Nonfundamental as they are additions to the language
- C++ permits definition of new types and *classes*
  - A class is a special kind of type
- Class objects typically have
  - *Data members* that represent attributes and values
  - *Member functions* for object inspection and manipulation
  - Members are accessed using the selection operator (.)

    ```
    j = s.size();
    ```
  - *Auxiliary* functions for other behaviors
- Libraries often provide special-purpose types and classes
- Programmers can also define their own types and classes

Examples

- Standard Template Library (STL) provides class `string`
- EzWindows library provides several graphical types and classes
  - `SimpleWindow` is a class for creating and manipulating window objects
  - `RectangleShape` is a class for creating and manipulating rectangle objects

Class string

- Class string
  - Used to represent a sequence of characters as a single object
- Some definitions

```
string Name = "Joanne";
string DecimalPoint = ".";
string empty = "";
string copy = Name;
string Question = '?';      // illegal
```

Nonfundamental Types

- To access a library use a preprocessor directive to add its definitions to your program file

```
#include <string>
```

- The using statement makes syntax less clumsy
  - Without it

```
std::string s = "Sharp";
std::string t = "Spiffy";
```

  - With it

```
using namespace std; // std contains string
string s = "Sharp";
string t = "Spiffy";
```

Compound Assignment

- C++ has a large set of operators for applying an operation to an object and then storing the result back into the object
- Examples

```
int i = 3;
i += 4;                          // i is now 7
cout << i << endl;

float a = 3.2;
a *= 2.0;                        // a is now 6.4
cout << a << endl;
```

Increment and Decrement

- C++ has special operators for incrementing or decrementing an object by one
- Examples

```
int k = 4;
++k;                          // k is 5
k++;                          // k is 6
cout << k << endl;
int i = k++;                  // i is 6, k is 7
cout << i << " " << k << endl;
int j = ++k;                  // j is 8, k is 8
cout << j << " " << k << endl;
```

Class string

- Some string member functions
  - size() determines number of characters in the string

    ```
    string Saying = "Rambling with Gambling";
    cout << Saying.size() << endl;       // 22
    ```

  - substr() determines a substring (Note first position has index 0)

    ```
    string Word = Saying.substr(9, 4); // with
    ```

  - find() computes the position of a subsequence

    ```
    int j = Saying.find("it");          // 10
    int k = Saying.find("its");         // ?
    ```

Class string

- Auxiliary functions and operators
  - getline() extracts the next input line

```
string Response;
cout << "Enter text: ";
getline(cin, Response, '\n');
cout << "Response is \"" << Response
 << "\"″ << endl;
```

  - Example run

```
Enter text: Want what you do
Response is "Want what you do"
```

Class string

- Auxiliary operators
  - + string concatenation

```
string Part1 = "Me";
string Part2 = " and ";
string Part3 = "You";
string All = Part1 + Part2 + Part3;
```

  - += compound concatenation assignment

```
string ThePlace = "Brooklyn";
ThePlace += ", NY";
```

```
#include <iostream>
using namespace std;
int main() {
   cout << "Enter the date in American format: "
    << "(e.g., January 1, 2001) : ";
      string Date;
   getline(cin, Date, '\n');
   int i = Date.find(" ");
   string Month = Date.substr(0, i);
   int k = Date.find(",");
   string Day = Date.substr(i + 1, k - i - 1);
   string Year = Date.substr(k + 2, Date.size()-k-2);
   string NewDate = Day + " " + Month + " " + Year;
   cout << "Original date: " << Date << endl;
   cout << "Converted date: " << NewDate << endl;
   return 0;
}
```