

Programming Fundamentals I

Chapter 2:

Basic Elements of C++

Dr. Adriana Badulescu

Objectives

- Become familiar with the basic components of a C++ program, including functions, special symbols, and identifiers
- Explore simple data types
- Discover how to use arithmetic operators
- Examine how a program evaluates arithmetic expressions
- Learn what an assignment statement is and what it does
- Become familiar with the string data type
- Discover how to input data into memory using input statements
- Become familiar with the use of increment and decrement operators
- Examine ways to output results using output statements
- Learn how to use preprocessor directives and why they are necessary
- Learn how to debug syntax errors
- Explore how to properly structure a program
- Learn how to write a C++ program

Introduction

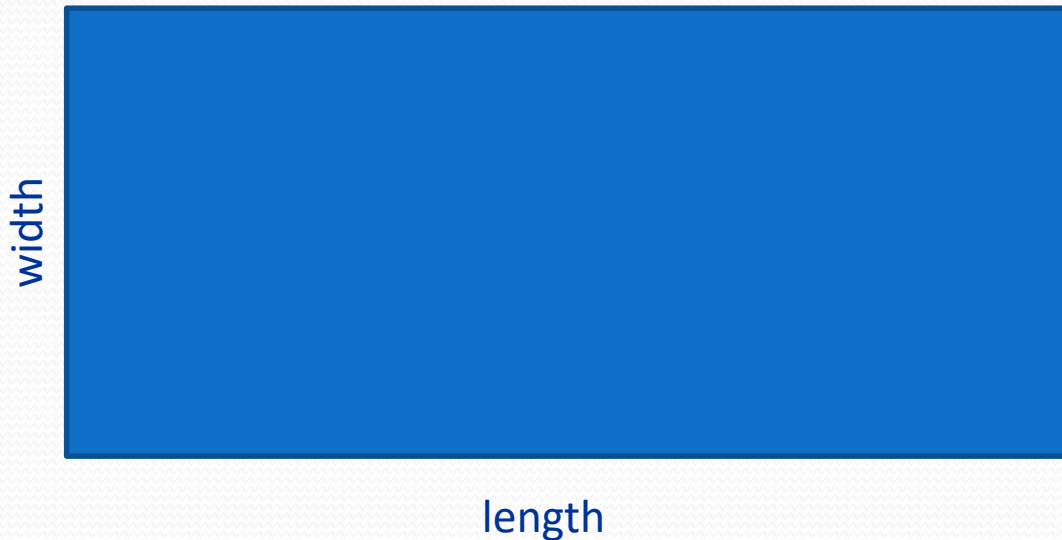
- **Computer program** - sequence of instructions or statements whose objective is to accomplish a task
- **Programming** - process of planning and creating a program
- **Programming language** - a set of rules, symbols, and special words used to communicate instructions to a computer



A C++ Program

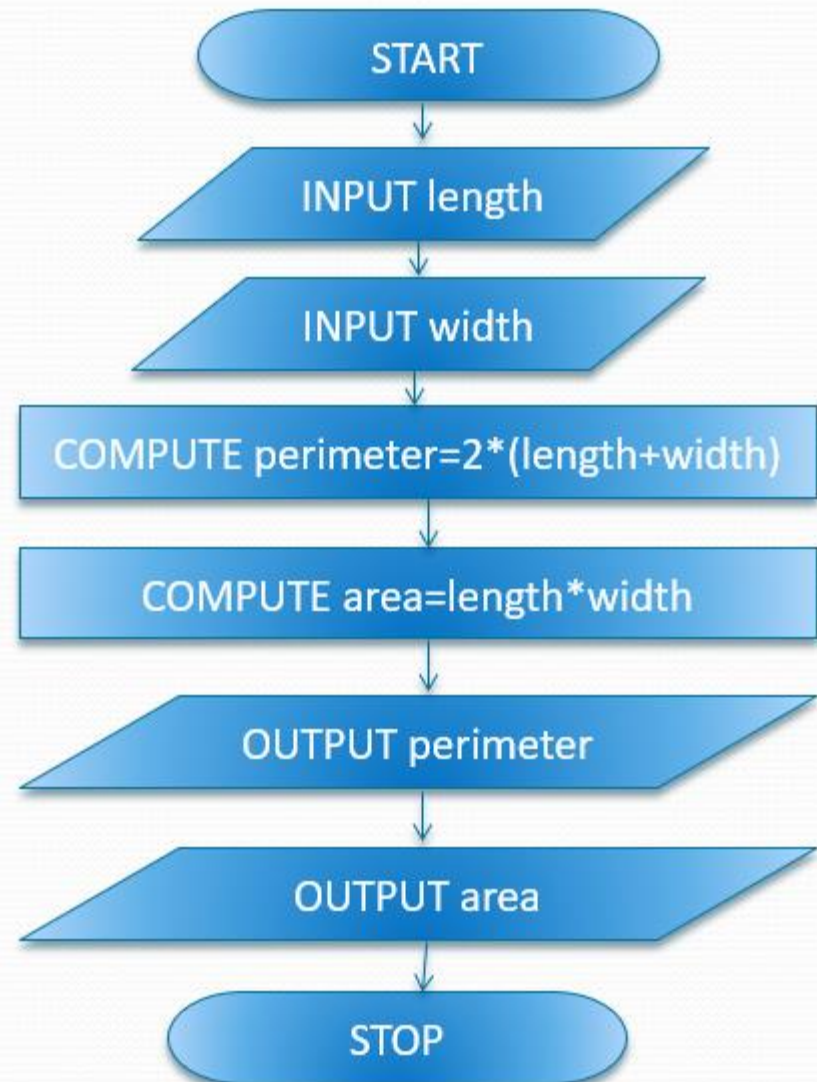
- **Problem:**

- Design an algorithm to find the perimeter and area of a rectangle



A C++ Program

- Flowchart



A C++ Program

■ Program

```
// Author: Adriana Badulescu
// Program: This program calculates the perimeter and area of a rectagle
// Input: The length and the width of the rectangle
// Output: The perimeter and the area of the rectangle

#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{
    // Tell the user what the program does
    cout << "This program calculates the perimiter and area of a rectangle.\n";
```

A C++ Program

```
//INPUT the length of the rectangle

// Declare variables for storing the length of the rectangle
int length;
// Prompt the user for the length
cout << "\nPlease enter the length your rectangle:" << endl;
// Get the length from the user
cin >> length;

//INPUT the width of the rectangle

// Declare variables for storing the width of the rectangle
int width;
// Prompt the user for the length
cout << "\nPlease enter the width your rectangle:" << endl;
// Get the width from the user
cin >> width;
```

A C++ Program

```
// COMPUTE the perimeter of the rectangle
int perimeter = 2 * (width + length);

// COMPUTE the area of the rectangle
int area = width * length;

//OUTPUT the perimeter
cout << "\nThe perimeter of your rectangle is: " << perimeter << endl;

//OUTPUT the area
cout << "\nThe area of your rectangle is: " << area << endl;

//Prevent the console from closing

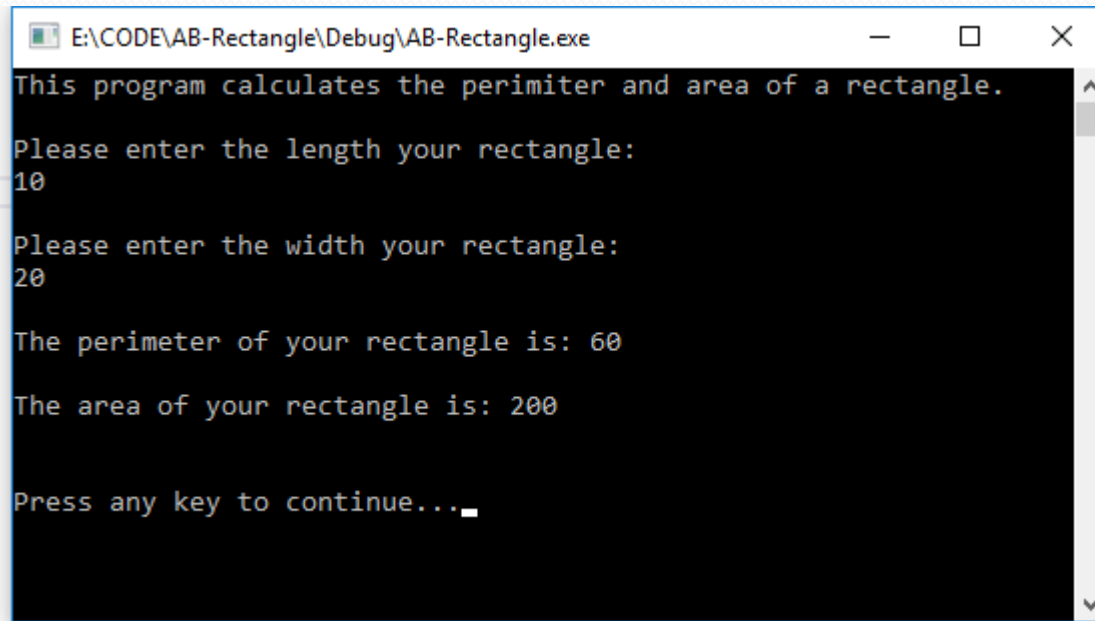
//Prompt the user to press any key
cout << "\n\nPress any key to continue...";

// Wait for a character before closing the console
_getch();

return 0;
}
```


A C++ Program

■ Sample Run/Output

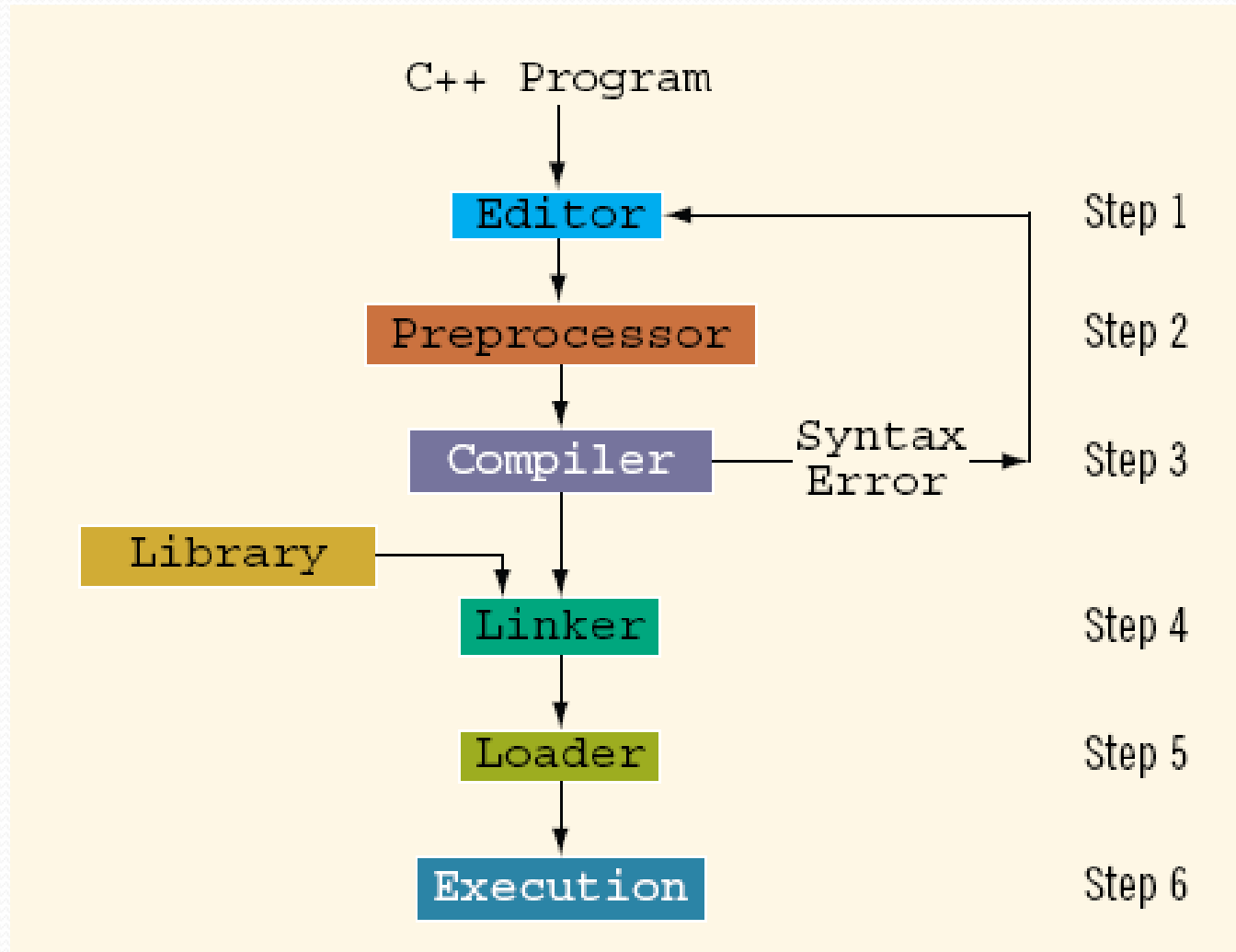
A screenshot of a Windows command prompt window titled "E:\CODE\AB-Rectangle\Debug\AB-Rectangle.exe". The window has a black background with white text. The text inside the window is as follows:

```
This program calculates the perimeter and area of a rectangle.  
Please enter the length your rectangle:  
10  
Please enter the width your rectangle:  
20  
The perimeter of your rectangle is: 60  
The area of your rectangle is: 200  
Press any key to continue..._
```

Processing a C++ Program

- To execute a C++ program:
 1. **Editor**: Use an editor to create a **source program** in C++
 2. **Preprocessor**: Include the code from the preprocessor directives
 3. **Compiler**: Use the compiler to:
 - Check that the program obeys the rules
 - Translate into machine language (**object program**)
 4. **Linker**: Combines object program with other programs and libraries provided by the software development kit (SDK) to create **executable code**
 5. **Loader**: Loads executable program into main memory
 6. **Execute** the program

Processing a C++ Program



Processing a C++ Program

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
```

Preprocessor directives
start with #

```
int main()
{
```

C++ Reserved word
(keywords) are in blue

```
// Write to console
```

```
cout << "My first C++ program." << endl;
```

Part of the `std` namespace

```
// Read a character (to prevent the  
console from closing before we can see the  
message)
```

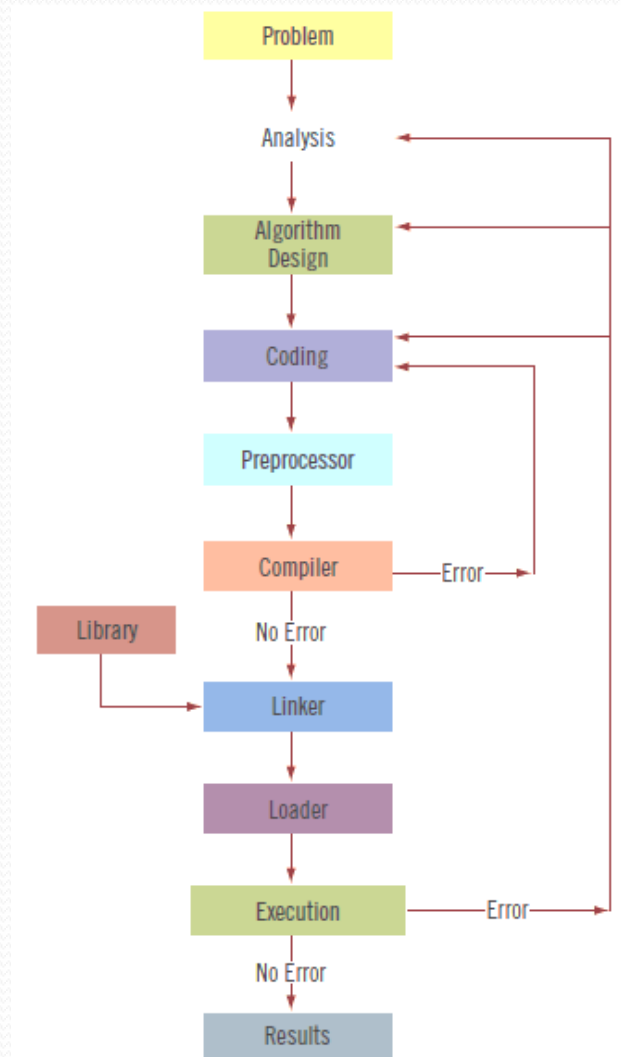
```
_getch();
```

```
return 0;
```

```
}
```

Comments start with `//`
are green and are
ignored by the compiler

The Problem Analysis–Coding–Execution Cycle



The Basics of a C++ Program

- **Function:** collection of statements; when executed, accomplishes something
 - May be predefined or standard
- **Syntax:** rules that specify which statements (instructions) are legal
- **Semantic rule:** meaning of the instruction

Comments

- **Comments** are for the reader, not the compiler
- **Two types:**

- **Single line:** begin with `//`

```
// This is a C++ program. It prints the sentence:  
// Welcome to C++ Programming.
```

- **Multiple line:** enclosed between `/*` and `*/`

```
/*  
    You can include comments that can  
    occupy several lines.  
*/
```

Whitespaces

- Every C++ program contains **whitespaces**
- Include blanks, tabs, and newline characters
- Used to separate special symbols, reserved words, and identifiers
- Can be used to make the program readable

```
int A, X1;  
int B, X2;  
int C, X3;  
int D, X4;
```


Whitespaces

```
// Program: This program calculates the perimeter and area of a rectangle
// Input: The length and the width of the rectangle
// Output: The perimeter and the area of the rectangle

#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{
    // Tell the user what the program does
    cout << "This program calculates the perimeter and area of a rectangle.\n\n";

    //INPUT the length of the rectangle
    // Declare variable for storing the length
    int length;
    // Prompt the user for the length/ tell the user to enter the length
    cout << "\nPlease enter the length: ";
    // Get the length from the user
    cin >> length;

    //INPUT the width of the rectangle
    // Declare variable for storing the width
    int width;
    // Tell the user to enter the width
    cout << "\nPlease enter the width: ";
    // Get the width from the user
    cin >> width;

    //COMPUTE the perimeter of the rectangle
    float perimeter = 2 * (width + length);

    //COMPUTE the area of the rectangle
    float area = width * length;

    //OUTPUT the perimeter
    cout << "\nThe perimeter of your rectangle is: " << perimeter << endl;

    //OUTPUT the area
    cout << "\nThe area of your rectangle is: " << area << endl;

    // Wait for a character before closing the console
    _getch();

    return 0;
}
```

tab

white space

new line

Tokens

- **Token:** the smallest individual unit of a program written in any language
- C++ tokens include
 - special symbols
 - word symbols
 - identifiers

Special Symbols and Reserved Words

- Special symbols

+	?
-	,
*	<=
/	!=
.	==
;	>=

- Reserved words, keywords, or word symbols

- Cannot be redefined within program

int	const
float	main
double	void
char	return

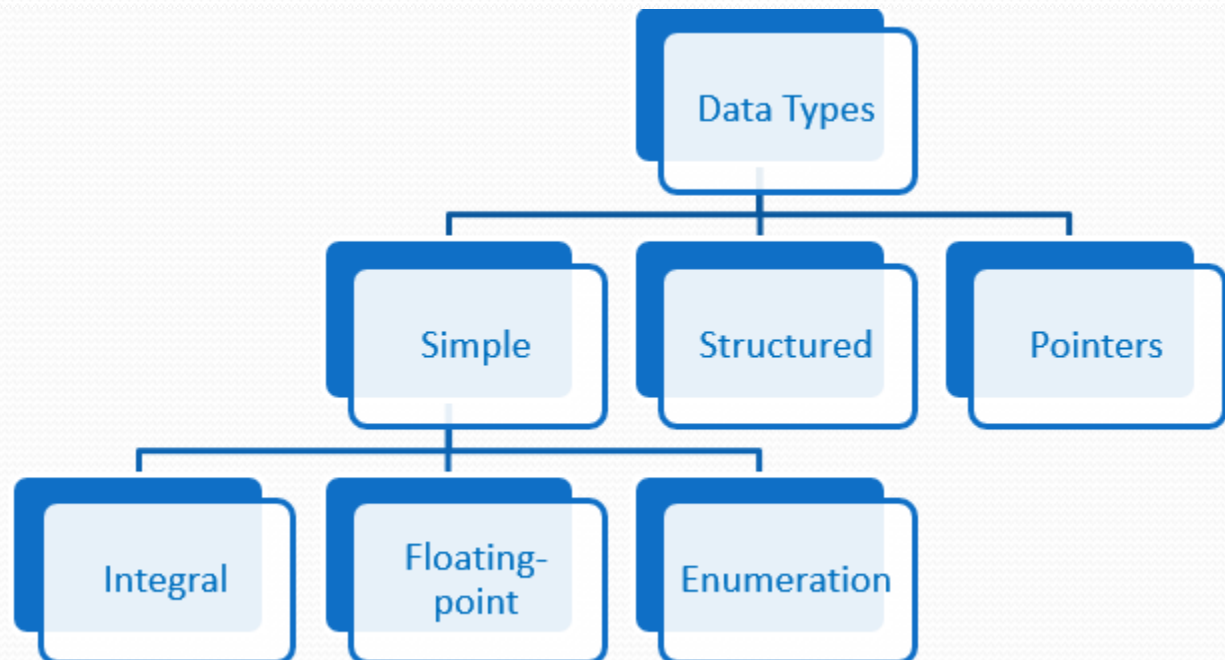
Identifiers

- Consist of letters, digits, and the underscore character ('_')
- Must begin with a letter or underscore
- C++ is case sensitive - `NUMBER` is not the same as `number` or `Number`
- May be redefined within program

Legal identifiers	Illegal identifiers
first conversion payRate _salary	Employee salary Hello! one+two 2nd

Data Types

- **Data type:** set of values together with a set of operations
- Different compilers may allow different ranges of values



Simple Data Types

- Three categories of simple data
 - **Integral**: integers (numbers without a decimal)
 - **Floating-point**: decimal numbers
 - **Enumeration type**: user-defined data type

Integral Data Types

	Data Type Name	Storage (in bytes)	Range of Values	
Logical values	bool	1	true and false	true or 1 false or 0
ASCII characters	Char	1	-128 to 127	'0' or 48 'A' or 65 'a' or 97
Positive and Negative numbers	short	2	-32,768 to 32,767	
	int	4	-2,147,483,648 to 2,147,483,647	-1000
	long	4	-2,147,483,648 to 2,147,483,647	0
	long long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	+10
Unsigned/ Positive numbers	unsigned char	1	0 to 255	
	unsigned short	2	0 to 65,535	
	unsigned int	4	0 to 4,294,967,295	1000
	unsigned long	4	0 to 4,294,967,295	0
	unsigned long long	8	0 to 18,446,744,073,709,551,615	+10

Floating-Point Data Types

- Represent real numbers

Real Number (floating-point notation)	Real-Numbers (scientific)
75.92	7.59200E1
0.18	1.80000E-1
-1.482	-1.4842000E0
7800.0	7.800000E3

- Types

Name	Storage (in bytes)	Range of Values	Example
<code>float</code>	4	3.4E +/- 38 (7 digits)	0.1234567 -5.8 0
<code>double</code>	8	1.7E +/- 308 (15 digits)	0.123456789012345 -1.236

Floating-Point Data Types

- **Precision** – the maximum number of significant digits (i.e. decimal places)

Data Type	Precision	Number of significant digits
float	Single precision	6 or 7 digits
double	Double precision	15 digits

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20			
A=	0	.	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0		
B=	0	.	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	0	7	8	9	0		
C=	0	.	1	2	3	4	5	6	7	0	9	0	1	2	3	4	5	6	7	8	9	0		
D=	0	.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
E=	0	.	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0		
F=	0	.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		
		float							double															

Arithmetic Operations

Operations	Symbol	Meaning	Example
addition	+	Adds 2 numbers	$5+3 \rightarrow 8$ $\text{'a'}+2 \rightarrow \text{'c'}$
subtraction	-	Subtracts 2 numbers	$5-3 \rightarrow 2$ $\text{'9'}-3 \rightarrow \text{'6'}$
multiplication	*	Multiplies 2 numbers	$5*3 \rightarrow 15$ $1.2*5.1 \rightarrow 6.12$
division	/	Divides 2 numbers (the results is the integral part of the division if numbers are integral or a floating-point otherwise)	$5/3 \rightarrow 1$ $10.5/4.2 \rightarrow 2.5$
modulus operator	%	The remainder from division	$5\%3 \rightarrow 2$

Order of Precedence

- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- When operators are on the same level
 - Performed from left to right (associativity)
- $3 * 7 - 6 + 2 * 5 / 4 + 6$ means
 $(((3 * 7) - 6) + ((2 * 5) / 4)) + 6$

Order of Precedence

$$9 + 8 - \underline{7 \% 6} + \underline{5 / 4} - \underline{3 * 2} / 1$$

$$9 + 8 - 1 + 1 - \underline{6} / 1$$

$$\underline{9 + 8 - 1} + 1 - 6$$

$$\underline{17 - 1} + 1 - 6$$

$$\underline{16} + 1 - 6$$

$$\underline{17} - 6$$

$$11$$

Order of Precedence

$$9 + 8 - (7 \% 6 + 5) / 4 - 3 * 2 / 1$$

$$9 + 8 - (1 + 5) / 4 - 3 * 2 / 1$$

$$9 + 8 - 6 / 4 - 3 * 2 / 1$$

$$9 + 8 - 1 - 6 / 1$$

$$9 + 8 - 1 - 6$$

$$17 - 1 - 6$$

$$16 - 6$$

$$10$$

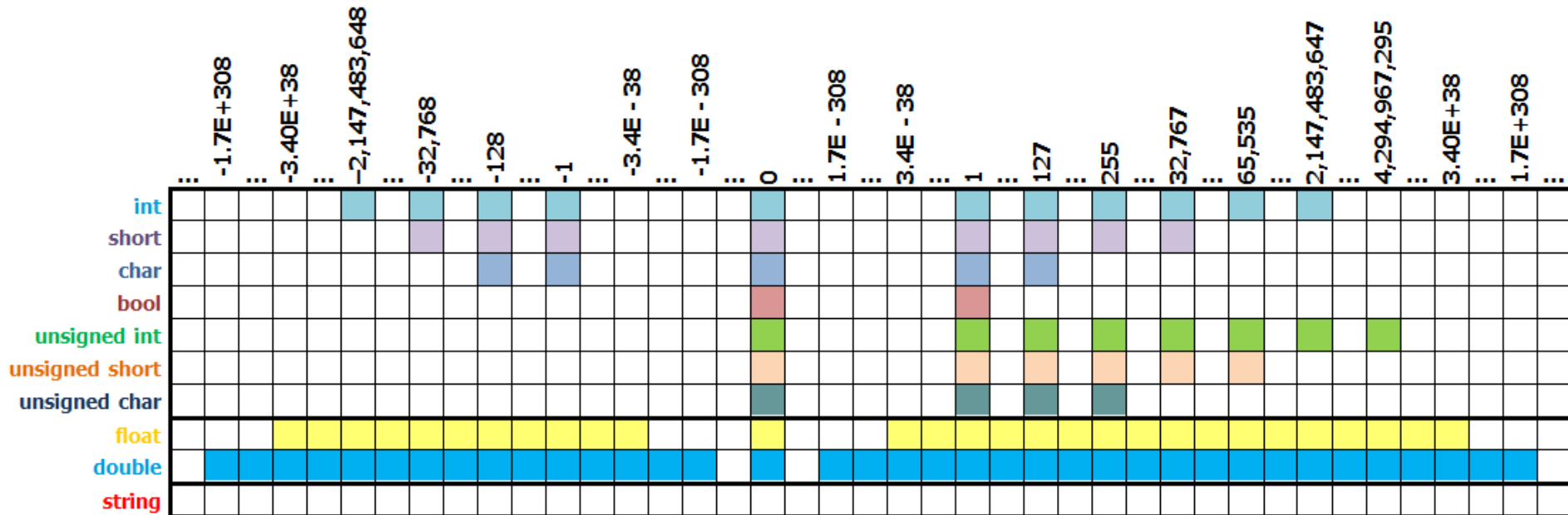
Expressions

- Integral expression contains only integer operands and yields an integral result
 - Example: $2 + 3 * 5$
- Floating-point expression contains only floating-point operands and yields a floating-point result
 - Example: $12.8 * 17.5 - 34.50$
- Mixed expression contains operands of different data types (integers and floating-point)
 - Example: $5.4 * 2 - 13.6 + 18/2 - 7/5$
 - Integers are promoted to floating-point

Type Conversions (Casting)

Promotion – from a smaller type to a compatible larger type

Typecasting – between 2 compatible types



Type Conversion (Casting)

- Implicit type coercion (promotion): when value of one type is automatically changed to another type

- `5.2/2=2.6` // 2 is promoted to float 2.0
- `1+3.5=4.5` // 1 is promoted to float 1.0

- Cast operator (typecasting): provides explicit type conversion

- static casting: `static_cast<dataTypeName>(expression)`
- C-like casting: `(dataTypeName) expression`
or `dataTypeName (expression)`

```
static_cast<int>(7.8+static_cast<double>(15)/2) -> 15
```

```
static_cast<int>(7.8+static_cast<double>(15/2)) -> 14
```


string Type

- A **string** is a sequence of zero or more characters
 - e.g. "John Doe", "Cat", "dog"
- Null or empty string: a string with no characters ""
- Each character has **relative position** in string
 - Position of first character is 0
- Length of a string is number of characters in it
 - e.g. length of "William Jacob" is 13
- Concatenation: adding a second string after another

0	1	2	3	0	1	2	3	4	0	1	2	3	4	5	6	7	8			
s1=	C	A	T	S	s2=	&	D	O	G	S	s1+s2=	C	A	T	S	&	D	O	G	S

Input

- Data must be loaded into main memory before it can be manipulated
- Storing data in memory is a two-step process:
 - Instruct computer to allocate memory
 - Include statements to put data into memory

Allocating Memory with Constants and Variables

- **Named constant:** memory location whose content cannot change during execution
 - Syntax: `const DataType Identifier = Value;`
 - Example: `const double PI = 3.141592;`
- **Variable:** memory location whose content may change during execution
 - Syntax:

```
DataType Identifier;  
DataType Identifier = Value;  
DataType Identifier, Identifier;
```
 - Example: `int x;`
`string name;`

Putting Data into Variables

- There are two ways to initialize (place data into) a variable:

```
int feet;
```

- By using the assignment statement

```
feet = 35;
```

- By using a read statement

```
cin >> feet;
```

Assignment Statement

- Syntax:

```
Variable = Expression;
```

- Expression is evaluated and its value is assigned to the variable on the left side
- In C++, = is called the **assignment operator**

```
int number1, number2;  
char ch;  
string str;  
number1=5;  
number1= number2*10;  
ch= 'A' ;  
str="Some text";
```

Assignment Statement

- **Increment operator (++)**: increment variable by 1

Operator	Equivalent	Example
++var	var=var+1	++x;
var++	var=var+1	y=x++;

- **Decrement operator (--)**: decrement variable by 1

Operator	Equivalent	Example
--var	var=var-1	--y;
var--	var=var-1	X--;

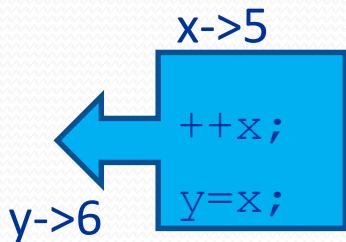
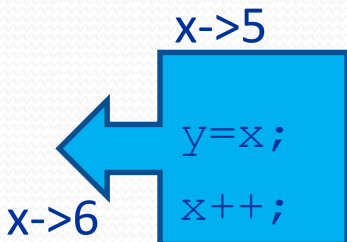
Assignment Statement

- **Compound assignments:** combination of assignment and arithmetic operator

Operator	Usage	Equivalent	Example
+=	var1+=var2	var1=var1+var2	x+=y;
-=	var1-=var2	var1=var1-var2	x-=5;
=	var1=var2	var1=var1*var2	Y*=1+4;
/=	var1/=var2	var1=var1/var2	x/=y;
%=	var1%=var2	var1=var1%var2	y%=9;

Assignment Statement

- What is the difference between a **Pre-increment** (`++variable`) and a **Post-increment** (`variable++`) operator?

Pre-increment	Post-increment
<pre>x = 5; y = ++x; cout << x << " " << y;</pre> <p>6 6</p> 	<pre>x = 5; y = x++; cout << x << " " << y;</pre> <p>6 5</p> 

Relational Operators

- Allow comparisons between two operands
- Evaluate to `true` or `false`

Operations	Symbol	Meaning	Example
Equal to	<code>==</code>	Return true if the numbers are equal	<code>3==9 -> false</code>
Not equal to	<code>!=</code>	Return true if the numbers are not equal	<code>3!=9 -> true</code> <code>'1'!=1 -> true</code>
Less than	<code><</code>	Return true if the first number is smaller than the second number	<code>3<9 -> true</code> <code>'A'<'a' -> true (65<97)</code>
Less than equal	<code><=</code>	Return true if the first number is smaller (or equal) than the second number	<code>3<=9 -> true</code> <code>2.5<=1.2 -> false</code>
Greater that	<code>></code>	Return true if the first number is larger than the second number	<code>3>3 -> false</code>
Greater than or equal to	<code>>=</code>	Return true if the first number is larger (or equal) than the second number	<code>3>=3 -> true</code> <code>"Hello">="Hi" -> false</code> <code>"Hello">"hello" -> false</code> <code>"Billy" >= "Bill" -> true</code>

Logical Operations

- For `bool` and other integral data type (`0` is `false`, anything else is `true`)

Operations	Symbol	Meaning	Example
negation	!	Negate the value (true becomes false and false becomes true)	!0 -> true (1) !1 -> false (0) !3 -> false (0)
and	&&	Return true if both values are true and both otherwise	1 && 0 -> false (0) 1 && 3 -> true (1) 0 && 0 -> false (0)
or		Return true if at least one of the values are true and both otherwise	1 0 -> true (1) 1 3 -> true (1) 0 0 -> false(0)

Order of Precedence of Common Operators

1. + (positive) , - (negative)
2. ++ (post-increment), -- (post-decrement)
3. ! (not), ++ (pre-increment), -- (pre-decrement)
4. * (multiplication), / (division), % (modulus)
5. + (addition), - (subtraction)
6. <, <=, >, >= (comparison)
7. == (equal-to), != (not-equal-to)
8. && (and)
9. || (or)
10. = (assignment), +=, -=, *=, /=, %= (compound assignment)

Order of Precedence

$$9 \ \&\& \ 8 == 7 < !6 \ * \ \underline{-5} / 4 + 3 \% 2 - 1$$

$$9 \ \&\& \ 8 == 7 < \underline{!6} \ * \ (-5) / 4 + 3 \% 2 - 1$$

$$9 \ \&\& \ 8 == 7 < \underline{0} \ * \ \underline{(-5)} / 4 + \underline{3 \% 2} - 1$$

$$9 \ \&\& \ 8 == 7 < \quad \underline{0} \quad / 4 + \quad 1 \quad - 1$$

$$9 \ \&\& \ 8 == 7 < \quad \quad \underline{0} \quad + \quad 1 \quad - 1$$

$$9 \ \&\& \ 8 == \underline{7 <} \quad \quad \underline{0}$$

$$\underline{9 \ \&\& \ 8 ==} \quad \underline{0}$$

$$\underline{9 \ \&\&} \quad \underline{0}$$

$$0$$

Input (Read) Statement

- `cin` is used with `>>` (the **stream extraction operator**) to gather input/read data

```
cin >> Variable;  
cin >> Variable1 >> Variable2 ... ;
```

- Example:

```
cin >> miles;  
cin >> feet >> inches;
```

Input (Read) Statement

- Depends of the type of variable you read into

Data Type	What does it reads?	Example
		tab - 1 2 3 . 4 5 A New line
int	Skips leading whitespaces read optional sign sequence of digits	tab - 1 2 3 . 4 5 A New line
char	Skips leading whitespaces read ASCII character	tab - 1 2 3 . 4 5 A New line
double	Skips leading whitespaces read optional sign sequence of digits optional decimal point sequence of digits	tab - 1 2 3 . 4 5 A New line
string	Skips leading whitespaces Reads ASCII characters until something else	tab - 1 2 3 . 4 5 A New line

Output

- `cout` and `<<` (the **stream insertion operator**) are used to output the value of an expression at the current cursor position or format the output

```
cout << Expression or Manipulator  
      << Expression2 or Manipulator2 ... ;
```

- A manipulator is used to format the output
 - `endl` causes the insertion point to move to beginning of next line

- Example:

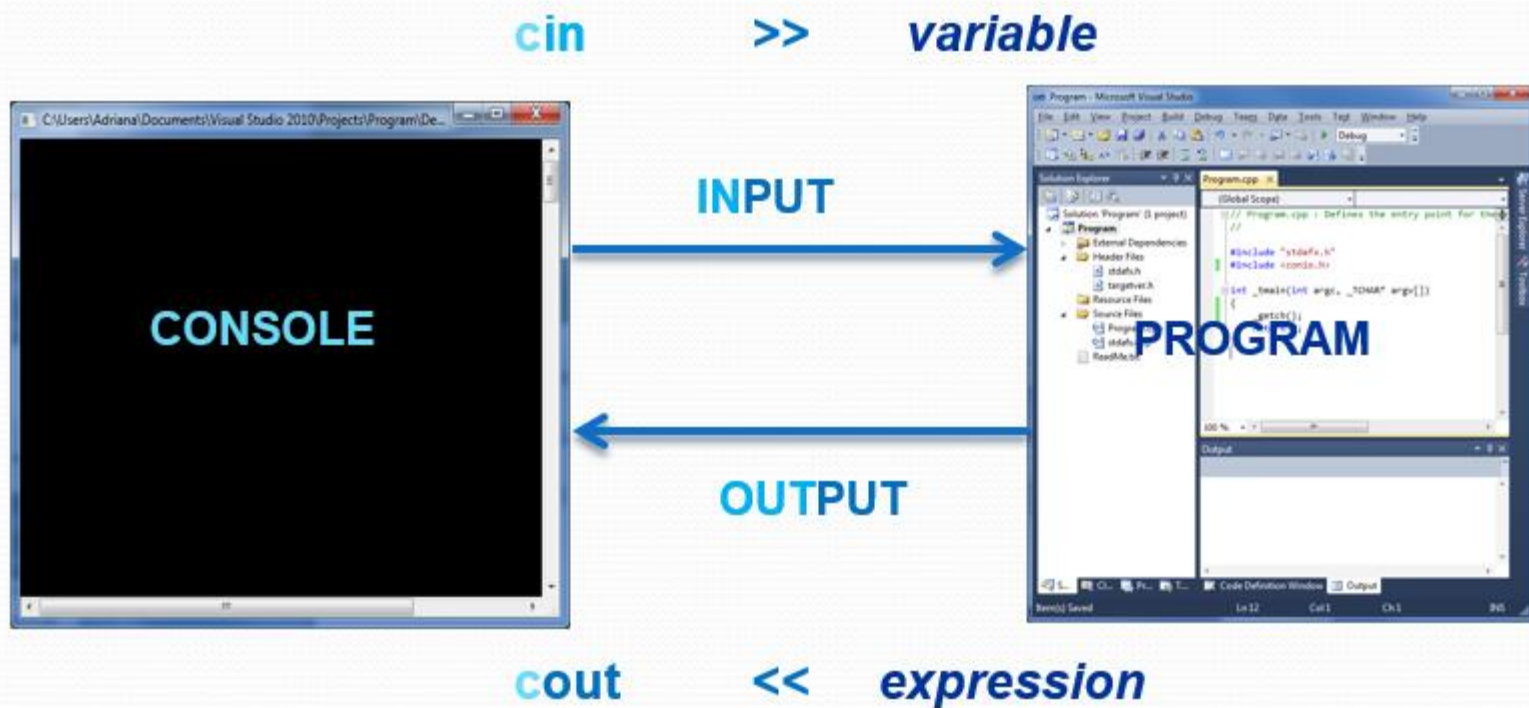
```
cout << "You entered the number N = " << N << endl;  
cout << " 29 / 4 = " << 29 / 4 << endl;
```

Output

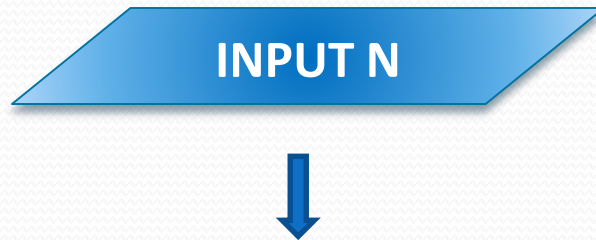
- **Escape sequences** are used to display characters that have a special meaning in C++ or in an output statement

Escape Sequence	Name	Description
\n	Newline	Cursor moves to the beginning of the next line
\t	Tab	Cursor moves to the next tab stop
\b	Backspace	Cursor moves one space to the left
\r	Return	Cursor moves to the beginning of the current line
\\	Backslash	Prints backslash
\'	Single quotation	Prints single quotation mark
\"	Double quotation	Prints single quotation mark

Input and Output



Input (Read) Statement



- `int N;`
- `N = 0;`
- `cout << "Enter an integral number N : ";`
- `cin >> N;`
- `cout << "You entered the number N = " << N;`

Input (Read) Statement

```
#include <iostream>

using namespace std;

int main()
{
    int feet;
    int inches;

    cout << "Enter two integers separated by one or more spaces: ";
    cin >> feet >> inches;
    cout << endl;

    cout << "Feet = " << feet << endl;
    cout << "Inches = " << inches << endl;

    return 0;
}
```

Sample Run: In this sample run, the user input is shaded.

Enter two integers separated by one or more spaces: 23 7

Feet = 23

Inches = 7

Preprocessor Directives

- C++ has a small number of operations
- Many functions and symbols needed are provided as collection of libraries
- Preprocessor directives are commands supplied to the preprocessor to include the header files for these libraries
- Syntax to include a header file:

```
#include <HeaderFileName>
```

- To use `cin`, `cout`, and any of their operations (that are declared in the header file `iostream` within the `std` namespace):

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

- To use the `string` type, you need to include

```
#include <string>
```

Creating a C++ Program

- A **C++ program** is a collection of functions, one of which is the function main
- The first line of the function main is called the heading of the function: `int main()`
- The statements enclosed between the curly braces (`{` and `}`) form the body of the function
 - Contains two types of statements:
 - Declaration statements: declare things, such as variables
 - Executable statements: perform calculations, manipulate data, create output, accept input, etc.

Creating a C++ Program

- Begin the program with comments for documentation
- Include header files
- Declare named constants, if any
- Write the definition of the function `main`

```
int main ()  
{  
    DeclarationStatements  
    ExecutableStatements  
    return 0;  
}
```

A Quick Look at a C++ Program

■ Program

```
// Author: Adriana Badulescu
// Program: This program calculates the perimeter and area of a rectagle
// Input: The length and the width of the rectangle
// Output: The perimeter and the area of the rectangle

#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{
    // Tell the user what the program does
    cout << "This program calculates the perimiter and area of a rectangle.\n";
```

A Quick Look at a C++ Program

```
//INPUT the length of the rectangle

// Declare variables for storing the length of the rectangle
int length;
// Prompt the user for the length
cout << "\nPlease enter the length your rectangle:" << endl;
// Get the length from the user
cin >> length;

//INPUT the width of the rectangle

// Declare variables for storing the width of the rectangle
int width;
// Prompt the user for the length
cout << "\nPlease enter the width your rectangle:" << endl;
// Get the width from the user
cin >> width;
```


A Quick Look at a C++ Program

```
// COMPUTE the perimeter of the rectangle
int perimeter = 2 * (width + length);

// COMPUTE the area of the rectangle
int area = width * length;

//OUTPUT the perimeter
cout << "\nThe perimeter of your rectangle is: " << perimeter << endl;

//OUTPUT the area
cout << "\nThe area of your rectangle is: " << area << endl;

//Prevent the console from closing

//Prompt the user to press any key
cout << "\n\nPress any key to continue...";

// Wait for a character before closing the console
_getch();

return 0;
```

Prevent the Console from Closing

- When you create a Visual C++ Windows Console, the Console windows is going to close when the program reaches the return statement
- To prevent that, you can make the program wait for a user to press a keyboard key before the console closing

```
#include <conio.h>
```

```
//prevent the console from closing
```

```
//prompt user to press any key
```

```
cout << "\n\nPress any key to exit...";
```

```
//read a key
```

```
_getch();
```

Program Style and Form

- Every C++ program has a function `main`
- Programs must also follow syntax rules
- Other rules serve the purpose of giving precise meaning to the language

Use of Whitespaces

- In C++, you use one or more blanks to separate numbers when data is input
 - Used to separate reserved words and identifiers from each other and from other symbols
 - Must never appear within a reserved word or identifier

Form and Style

- Consider two ways of declaring variables:

- Method 1

```
int feet, inch;  
double x, y;
```

- Method 2

```
int feet, inch; double x, y;
```

- Both are correct; however, the second is hard to read

Documentation

- A well-documented program is easier to understand and modify
- You should use comments to document programs
- Comments should appear in a program to:
 - Explain the purpose of the program
 - Identify who wrote it
 - Explain the purpose of particular statements

Prompt Lines

- **Prompt lines:** executable statements that inform the user what to do

```
cout << "Please enter a number between 1 and 10 and "  
      << "press the return key" << endl;  
cin >> num;
```

- Every input, should have a prompt message

Naming Identifiers

- Identifiers can be self-documenting:
 - `CENTIMETERS_PER_INCH`
- Avoid run-together words :
 - `annualsale`
 - Solution:
 - Capitalize the beginning of each new word:
`annualSale`
 - Inserting an underscore just before a new word:
`annual_sale`

Use of Semicolons, Brackets, and Commas

- All C++ statements end with a semicolon ‘;’
 - Also called a statement terminator
- { and } are not C++ statements
- Commas separate items in a list

Debugging - Understanding and Fixing Syntax Errors

- Compile a program
 - Compiler will identify the syntax error
 - Specifies the line numbers where the errors occur

```
Example2_Syntax_Errors.cpp
```

```
c:\chapter 2 source code\example2_syntax_errors.cpp(9) :  
    error C2146: syntax error :  
        missing ';' before identifier 'num'
```

```
c:\chapter 2 source code\example2_syntax_errors.cpp(11) :  
    error C2065: 'tempNum' :  
        undeclared identifier
```

- Learn how to spot and fix syntax errors

Syntax

- Errors in syntax are found in compilation

<code>int main ({</code>	Missing closed bracket “)”
<code>Int x;</code>	Unknown type (it should be int, not Char)
<code>float y</code>	Missing semicolon
<code>const double x=5;</code>	Redeclared variable (x)
<code>string string s;</code>	Invalid identifier (string is a reserved word)
<code>Char const;</code>	Invalid identifier (const is a reserved word)
<code>y = w + x;</code>	Unknown identifier (w)

Syntax

■ Errors in syntax are found in compilation

<code>x=y+s;</code>	Incorrect operation (string + float)
<code>y = 2*s;</code>	Incorrect operation (multiplying a string)
<code>x = y + x;</code>	Assigning value to a constant
<code>float y=5/0;</code>	Division by 0
<code>short l=32000+2000;</code>	Overflow (integer too large for data type)
<code>float z=1.5e-144;</code>	Underflow (floating-point interpreted as 0)
<code>float x=3.9e10+500.0</code>	Floating-point rounding error (reached maximum precision)
	Missing curly bracket “}”

Semantics

- Possible to remove all syntax errors in a program and still not have it run
- Even if it runs, it may still not do what you meant it to do
- For example,

$2 + 3 * 5$ and $(2 + 3) * 5$

are both syntactically correct expressions, but have different meanings

Example: Convert Length

- Write a program that takes as input a given length expressed in feet and inches and converts and outputs the length in centimeters
- Input: length in feet and inches
- Output: equivalent length in centimeters
- Process:
 - converts the length (from feet and inches) to total inches: multiply the feet by 12 and add the inches
 - Convert the length from total inches to centimeters: multiply the length in total inches by 2.54 (one inch is equal to 2.54 centimeters)

Example: Convert Length

■ Variables

```
int feet;           //variable to hold given feet
int inches;         //variable to hold given inches
int totalInches;    //variable to hold total inches
double centimeters; //variable to hold length in
                    //centimeters
```

■ Named Constant

```
const double CENTIMETERS_PER_INCH = 2.54;
const int INCHES_PER_FOOT = 12;
```

Example: Convert Length

- Prompt user for input
- Get data
- Echo the input (output the input)
- Find length in inches
- Output length in inches
- Convert length to centimeters
- Output length in centimeters

Example: Convert Length

```
// The problem convert a length in feet and inches to centimeters
// Input: length in feet and inches
// Output: length in centimeters

//header file
#include "stdafx.h"
#include <iostream>
#include <conio.h>

using namespace std;

//named constants
const double CENTIMETERS_PER_INCH = 2.54;
const int INCHES_PER_FOOT = 12;

int main ()
{
    //declare variables
    int feet, inches;
    int totalInches;
    double centimeter;

    //read the length in feet and inches
    cout << "Enter two integers, one for feet and one for inches: " << endl;
    cin >> feet >> inches;
    cout << endl;

    //write the input values
    cout << "You entered the length: " << feet << " feet and " << inches << " inches " << endl;
```

Example: Convert Length

```
//compute the length in total inches
totalInches = INCHES_PER_FOOT * feet + inches;

//write the length in total inches
cout << "\nThe total number of inches is " << totalInches << endl;

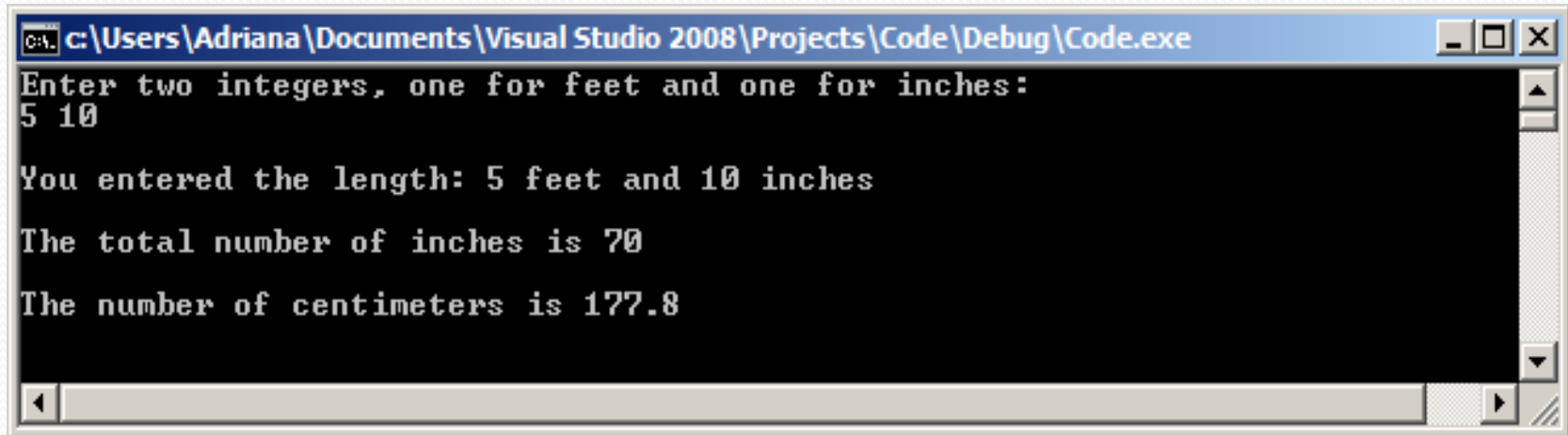
//compute the length in centimeters
centimeter = CENTIMETERS_PER_INCH * totalInches;

//write the length in centimeters
cout << "\nThe number of centimeters is " << centimeter << endl;

//wait for a character to prevent the console from closing
_getch();

return 0;
}
```

Example: Convert Length



A screenshot of a Windows command prompt window. The title bar shows the file path: `c:\Users\Adriana\Documents\Visual Studio 2008\Projects\Code\Debug\Code.exe`. The window contains the following text:

```
Enter two integers, one for feet and one for inches:  
5 10  
  
You entered the length: 5 feet and 10 inches  
  
The total number of inches is 70  
  
The number of centimeters is 177.8
```

Example: Make Change

- Program that takes as input any change expressed in cents and computes the number of half-dollars, quarters, dimes, nickels, and pennies to be returned, returning as many half-dollars as possible, then quarters, dimes, nickels, and pennies (in that order).
- Input: change in cents
- Output: equivalent change in half-dollars, quarters, dimes, nickels, and pennies

Example: Make Change

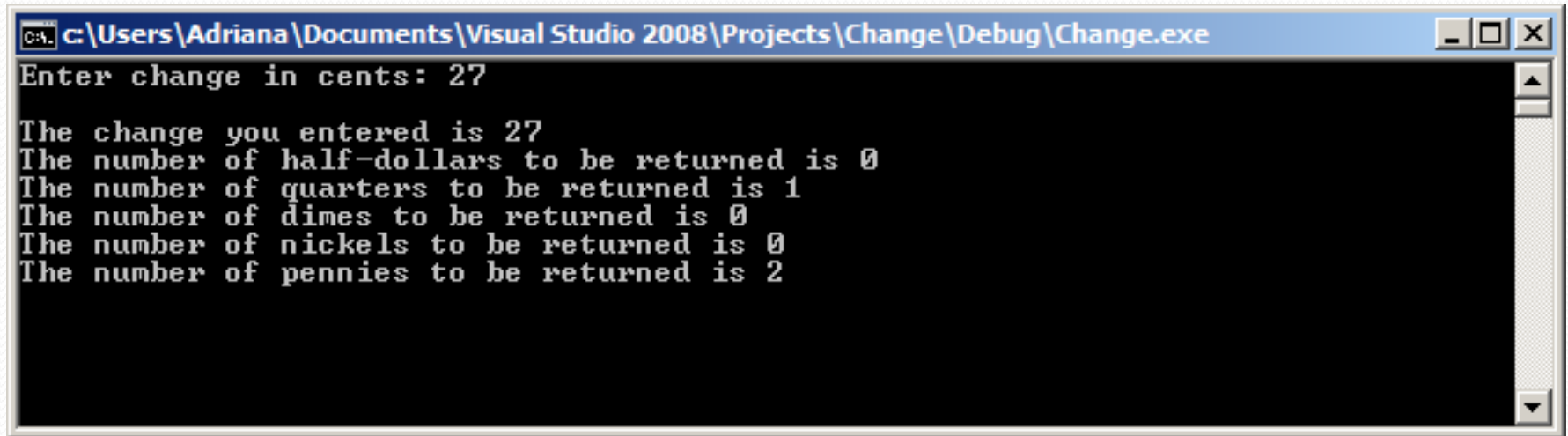
1. Get the change in cents
2. Compute number of half-dollars: $\text{Change} / 50$
3. Compute the remainder change: $\text{Change} = \text{Change} \% 50$
4. Compute number of quarters: $\text{Change} / 25$
5. Compute the remainder change: $\text{Change} = \text{Change} \% 25$
6. Compute number of dimes: $\text{Change} / 10$
7. Compute the remainder change: $\text{Change} = \text{Change} \% 10$
8. Compute number of nickels: $\text{Change} / 5$
9. Compute the number of pennies (remainder change):
 $\text{Change} \% 5$

Example: Make Change

```
#include <iostream>
#include <conio.h>
using namespace std;
const int HALF_DOLLAR = 50;
const int QUARTER = 25;
const int DIME = 10;
const int NICKEL = 5;
int main()
{
    int change;
    cout << "Enter change in cents: ";
    cin >> change;
    cout << endl;
    cout << "The change you entered is " << change << endl;
    cout << "The number of half-dollars to be returned is " << change / HALF_DOLLAR << endl;
    change = change % HALF_DOLLAR;
    cout << "The number of quarters to be returned is " << change / QUARTER << endl;
    change = change % QUARTER;
    cout << "The number of dimes to be returned is " << change / DIME << endl;
    change = change % DIME;
    cout << "The number of nickels to be returned is " << change / NICKEL << endl;
    change = change % NICKEL;
    cout << "The number of pennies to be returned is " << change << endl;
    _getch();
    return 0;
}
```

- Your code should have comments for every line of code

Example: Make Change



```
c:\Users\Adriana\Documents\Visual Studio 2008\Projects\Change\Debug\Change.exe
Enter change in cents: 27

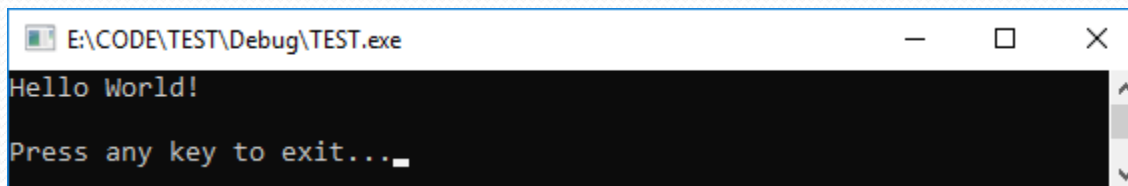
The change you entered is 27
The number of half-dollars to be returned is 0
The number of quarters to be returned is 1
The number of dimes to be returned is 0
The number of nickels to be returned is 0
The number of pennies to be returned is 2
```

Hello World

```
// MyFirstProgram.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
//THE LIBRARIES/PREPROCESSOR DIRECTIVES GO HERE!
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    //THE CODE GOES HERE!
    //START
    //OUTPUT "Hello World!"
    cout << "Hello World!";
    //STOP
    //prevent the console from closing
    //prompt the user to press a key
    cout << "\n\nPress any key to exit...";
    //read a key
    _getch();

    return 0;
}
```



Exercises with Numbers

- Read an integral number from the user into a variable called N1 and print it

//print a message to tell the user what the program does

```
cout << "This program reads an integral number N1 from the user and  
prints it";
```

//INPUT N1

//prompt the user for the number

```
cout << "\n\nPlease enter an integral number N1: ";
```

//allocate memory aka declare variable

```
int N1;
```

//put data into memory aka read into variable

```
cin >> N1;
```

//OUTPUT N1

```
cout << "The number N1 is "<<N1;
```

```
This program reads an integral number N1 from the user and prints it
```

```
Please enter an integral number N1: 10
```

```
The number N1 is 10
```

Exercises with Numbers

- Read another integral number from the user into a variable called N2 and print it

```
//print a message to tell the user what the program does
cout << "\n\nThis program reads another integral number N2 from the user
and prints it";
//INPUT N2
//prompt the user for the number
cout << "\n\nPlease enter an integral number N2: ";
//allocate memory aka declare variable
int N2;
//put data into memory aka read into variable
cin >> N2;
//OUTPUT N2
cout << "The number N2 is " << N2;
```

```
This program reads another integral number N2 from the user and prints it

Please enter an integral number N2: 2
The number N2 is 2
```

Exercises with Numbers

- Compute the sum of N1 and N2 and store it in a variable Sum and print the Sum

```
//COMPUTE Sum=N1+N2
//allocate memory aka declare variable
int Sum;
//put data into variable aka assign value to variable
Sum = N1 + N2;
//OUTPUT the Sum
cout << "\n\nThe sum of N1 and N2 is " << Sum;
```

```
The sum of N1 and N2 is 12
```

Exercises with Numbers

- Compute the sum of N1 and N2 and store it in a variable Sum and print the Sum

```
//COMPUTE Div=N1/N2
//allocate memory aka declare variable
float Div;
//put data into variable aka assign value to variable
Div = static_cast<float>(N1)/N2;
//OUTPUT Div
cout << "\n\nThe quotient of N1 and N2 is " << Div;
```

```
The quotient of N1 and N2 is 5
```

Exercises with Numbers

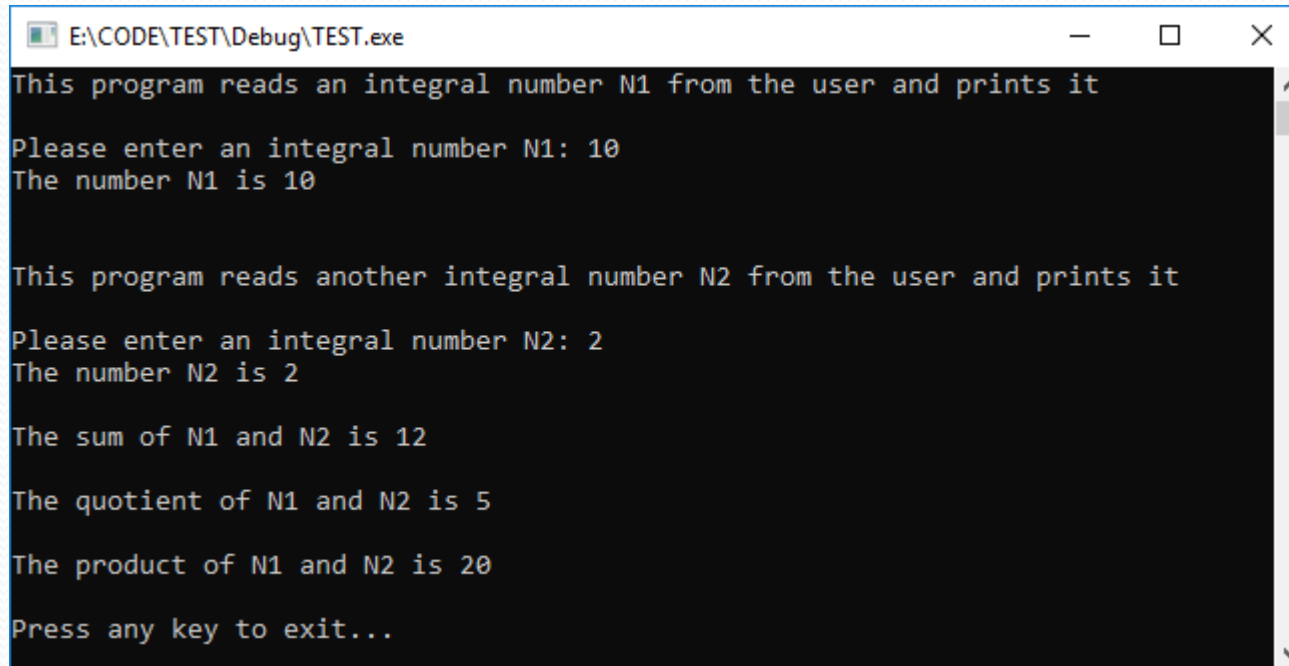
- Product of N1 and N2

//Compute the product of N1 and N2 in the output

```
cout << "\n\nThe product of N1 and N2 is " << (N1*N2);
```

```
The product of N1 and N2 is 20
```

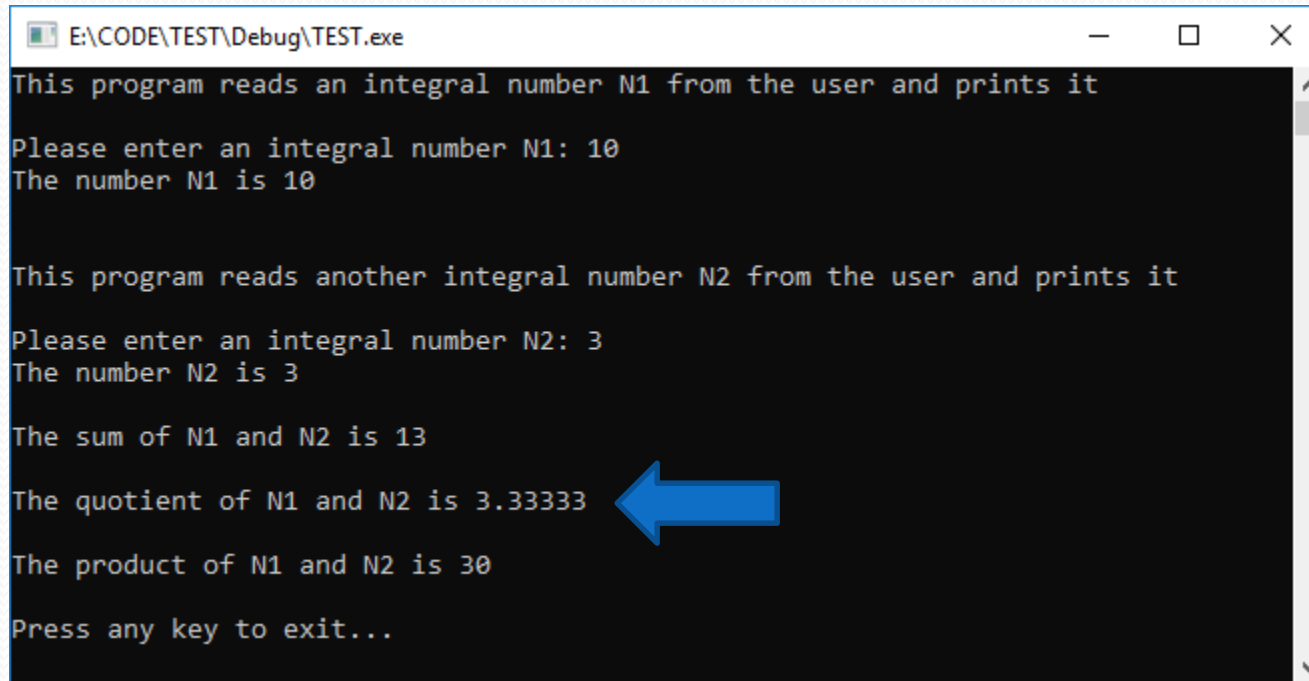
Exercises with Numbers



The screenshot shows a Windows command prompt window titled "E:\CODE\TEST\Debug\TEST.exe". The window contains the following text:

```
This program reads an integral number N1 from the user and prints it  
Please enter an integral number N1: 10  
The number N1 is 10  
  
This program reads another integral number N2 from the user and prints it  
Please enter an integral number N2: 2  
The number N2 is 2  
  
The sum of N1 and N2 is 12  
The quotient of N1 and N2 is 5  
The product of N1 and N2 is 20  
Press any key to exit...
```

Exercises with Numbers



```
E:\CODE\TEST\Debug\TEST.exe
This program reads an integral number N1 from the user and prints it
Please enter an integral number N1: 10
The number N1 is 10

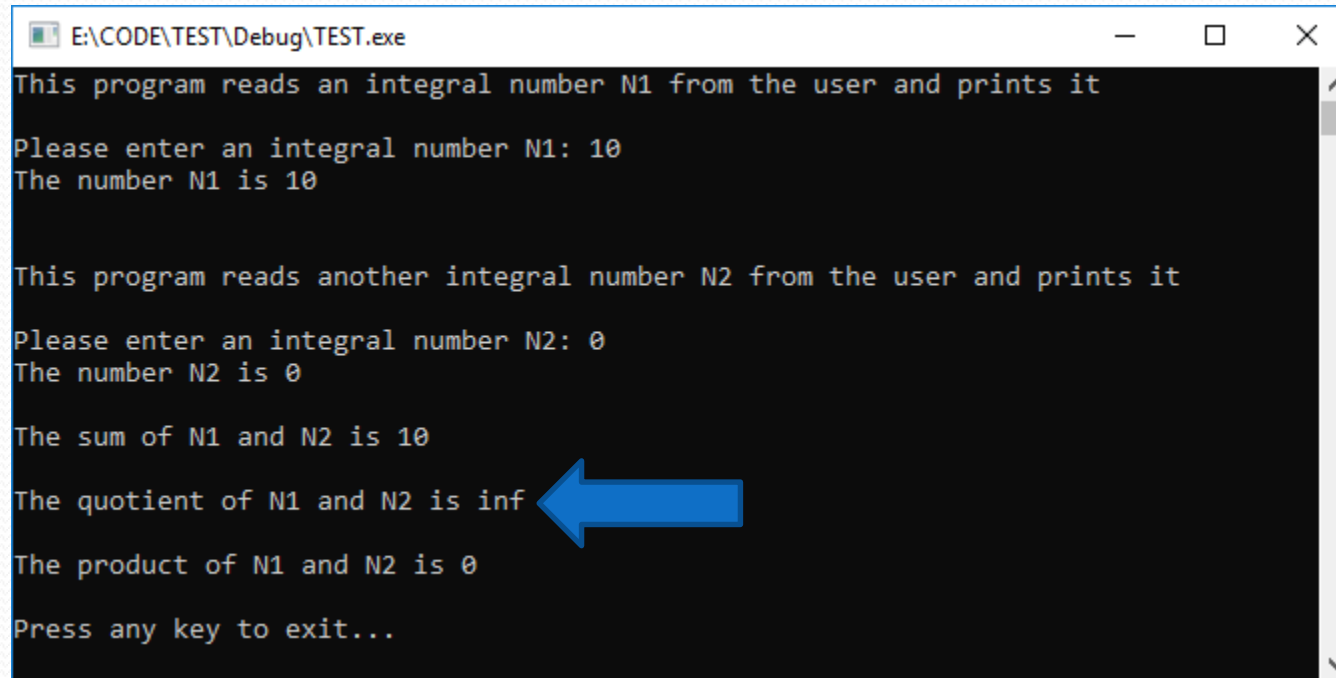
This program reads another integral number N2 from the user and prints it
Please enter an integral number N2: 3
The number N2 is 3

The sum of N1 and N2 is 13
The quotient of N1 and N2 is 3.33333
The product of N1 and N2 is 30

Press any key to exit...
```

- By default it is going to output all the decimals available for that type.

Exercises with Numbers



```
E:\CODE\TEST\Debug\TEST.exe
This program reads an integral number N1 from the user and prints it
Please enter an integral number N1: 10
The number N1 is 10

This program reads another integral number N2 from the user and prints it
Please enter an integral number N2: 0
The number N2 is 0

The sum of N1 and N2 is 10
The quotient of N1 and N2 is inf
The product of N1 and N2 is 0
Press any key to exit...
```

- We did not solve the problem! If N2 is 0, it cannot divide by 0, so, it output a bad value

Exercises with Strings

- Write a program that outputs "Hello USER! My name is PROGRAMMER!", replace PROGRAMMER with the programmer's name

```
cout << "This program outputs \"Hello USER! My name is  
PROGRAMMER!\"";
```

```
//OUTPUT "Hello USER! My name is PROGRAMMER!"  
cout << "\n\nHello USER! My name is Dr. Badulescu!";
```

```
This program outputs "Hello USER! My name is PROGRAMMER!"  
Hello USER! My name is Dr. Badulescu!
```

Summary

- C++ program structure
- Identifiers
- Arithmetic operators
- Arithmetic expressions
- Mixed expressions
- Type casting
- Constants and variables
- cin and stream extraction operator >>
- cout and stream insertion operator <<
- Preprocessor commands