Programming Fundamentals I

Chapter 3: Input/Output

Objectives

- Learn what a stream is and examine input and output streams
- Explore how to read data from the standard input device
- Learn how to use predefined functions in a program
- Explore how to use the input stream functions
- Become familiar with input failure
- Learn how to write data to the standard output device
- Discover how to use manipulators in a program to format output
- Learn how to perform input and output operations with the string data type
- Learn how to debug logic errors
- Become familiar with file input and output

I/O Streams and Standard I/O Devices

- **Stream**: sequence (stream) of bytes from source to destination
  - Bytes are usually characters, unless program requires other types of information
- **Input stream**: sequence of characters from an input device to the computer
- **Output stream**: sequence of characters from the computer to an output device

I/O Streams and Standard I/O Devices

- Use `iostream` header file to extract (receive) data from keyboard and send output to the screen

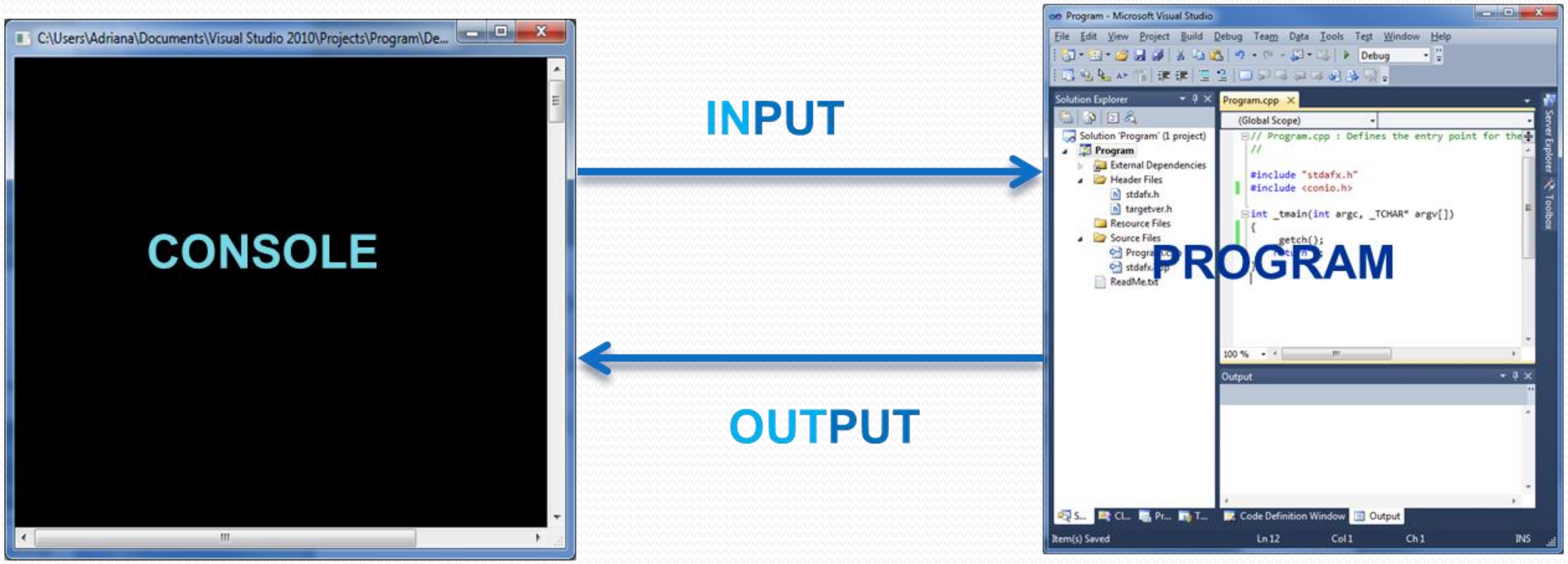
```
#include <iostream>
```

  - Contains definitions of two data types:
    - `istream`: input stream
    - `ostream`: output stream
  - Has two variables:
    - `cin`: stands for common input
    - `cout`: stands for common output

```
istream cin;
ostream cout;
```

I/O Streams and Standard I/O Devices

cin >> *variable*



```
// Program.cpp : Defines the entry point for the
//

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

int _tmain(int argc, _TCHAR* argv[])
{
    getch();
```

cout << *expression*

`cin` and the Extraction Operator >>

- The syntax of an input statement using **`cin`** and the extraction operator **>>** is:

```
cin >> Variable ;
cin >> Variable >> Variable … ;
```

- The extraction operator **>>** is binary
  - Left-side operand is an input stream variable (`cin`)
  - Right-side operand is a variable
- When scanning, **>>** skips all leading whitespace (blanks and certain nonprintable characters)

`cin` and the Extraction Operator >>

- >> read different values depending on the type of the right-side operand variable

Input	Statement	Value Stored in Variable/Memory

1	`int n;`
	`cin >> n;`
		`n = 1`

	`char c;`
	`cin >> c;`
		`c = '1'`

	`bool b;`
	`cin >> b;`
		`b = true`

	`float f;`
	`cin >> f;`
		`f = 1.0`

	`string s;`
	`cin>> s;`
		`s = "1"`

`cin` and the Extraction Operator >>

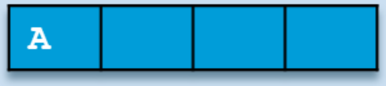
- Depends of the type of variable you read into

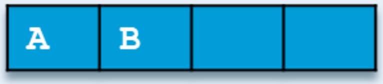
Data Type	What does it reads?	Example tab - 1 2 3 . 4 5 A New line
`int`	Skips leading whitespaces read optional sign sequence of digits	tab **-** **1** **2** **3** . 4 5 A New line
`char`	Skips leading whitespaces read ASCII character	tab **-** 1 2 3 . 4 5 A New line
`double`	Skips leading whitespaces 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

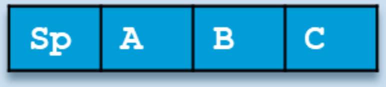
`cin` and the Extraction Operator >>

- When reading data into a `char` variable
  - **>>** skips leading whitespace, finds and stores only the next character
  - Reading stops after a single character

Statement	Input	Value Stored in Variable/Memory

`char c;`
`cin >> c;`
		`c = 'A'`

`char c;`
`cin >> c;`
		`c = 'A'`
		`'B' is held for later input`

`char c;`
`cin >> c;`
		`Skips leading whitespaces`
		`c = 'A'`
		`"BC" is held for later input`

`cin` and the Extraction Operator >>

- To read data into an integral variable (short, int, long, long long)
  - **>>** skips leading whitespace, reads + or - sign (if any), reads the digits until a non-digit character

Statement	Input	Value Stored in Variable/Memory

`int n;`
`cin >> n;`
	1	`n = 1`

`int n;`
`cin >> n;`
	1 2 . 3	`n = 12`
		`".3" is held for later input`

`int n;`
`cin >> n;`
	sp 1 2 C	`Skips leading whitespaces`
		`n = 12`
		`"C" is held for later input`

- Entering a char value into an int or double variable causes serious errors, called **input failure**

`cin` and the Extraction Operator >>

- To read data into a **floating point** `(float or double)` variable
  - **>>** skips leading whitespace, reads + or - sign (if any), reads the digits (including decimal point)
  - Reading stops on whitespace or non-digit character

Statement	Input	Value Stored in Variable/Memory

`double z;`
`cin >> z;`
	1 2 . 3	`z = 12.3`

`double z;`
`cin >> z;`
	1 2	`z = 12.0`

`double z;`
`cin >> z;`
	sp 1 2 . 3 4 A	Skips leading whitespaces
		`n = 12.34`
		"A" is held for later input

`cin` and the Extraction Operator >>

Statement	Input	Value Stored in Variable/Memory

`cin >> z`
`>> i`
`>> c;`
	1 | 2 | . | 3 | sp | 4 | sp | 5 | new line	`z = 12.3`
		`i = 4`
		`c = '5'`

`cin >> z`
`>> i`
`>> c;`
	1 | 2 | . | 3 | tab | 4 | new line
	5 | | | | | |
		`z = 12.3`
		`i = 4`
		`c = '5'`

`cin >> i`
`>> z`
`>> c;`
	1 | 2 | . | 3 | 4 | A | 5 | sp |	`i = 12`
		`z = .34`
		`c = 'A'`
		`"5 " is held for later input`

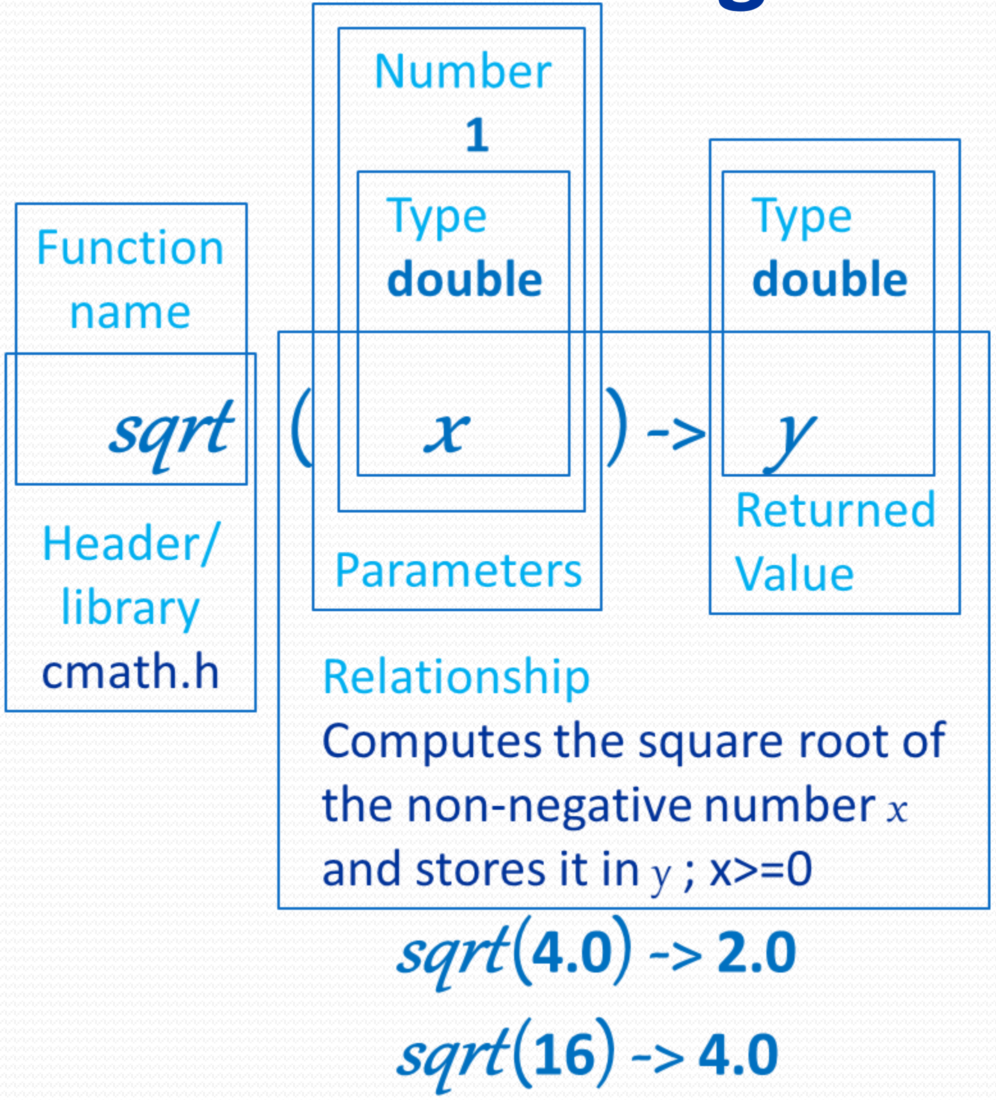
`cin >> z`
`>> c`
`>> i;`
	1 | 2 | . | 3 | 4 | A | sp | |	`z = 12.345`
		`c = 'A'`
		`computer waits for an input value for i`

Using Predefined Functions in a Program

- **Function (subprogram)**: set of instructions
  - When activated, it accomplishes a task
- `main` executes when a program is run
- Other functions execute only when called
- C++ includes a wealth of functions
- **Predefined functions** are organized as a collection of libraries called **header files** that may contain several functions

Using Predefined Functions in a Program

- To use a predefined function, you need :
  - Function name
  - What the function does / relationship
  - Name of the header file
  - Number of parameters
  - Order of parameters
  - Type of each parameter
  - Returned value



`cin` and the `get` Function

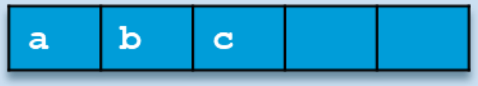
- The `get` function
  - Inputs next character (including whitespace)
  - Stores in memory location indicated by its argument
- The syntax of `cin` and the `get` function:

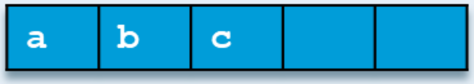
```
cin.get(VarChar);
```

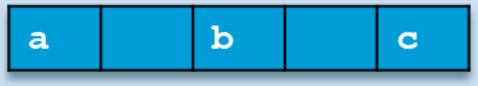
  - `VarChar` is a `char` variable
    - Is the argument (parameter) of the function
  - Read the next character and store it in `VarChar`
  - `cin.get(ch);`

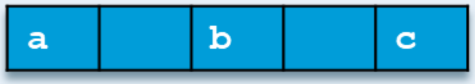
cin and the get Function

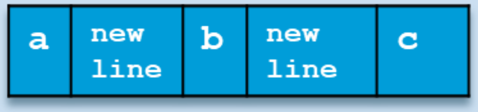
Statement	Input	Value Stored in Variable

cin>>ch1
>>ch2
>>ch3;
		ch1='a'
		ch2='b'
		ch3='c'

cin.get(ch1);
cin.get(ch2);
cin.get(ch3);
		ch1='a'
		ch2='b'
		ch3='c'

cin>>ch1
>>ch2
>>ch3;
		ch1='a'
		ch2='b'
		ch3='c'

cin.get(ch1);
cin.get(ch2);
cin.get(ch3);
		ch1='a'
		ch2=' '
		ch3= 'b'
		" c" is held for later input

cin.get(ch1);
cin.get(ch2);
cin.get(ch3);
		ch1='a'
		ch2=' \n'
		ch3= 'b'

The Dot Notation Between I/O Stream Variables and I/O Functions: A Precaution

- In the statement

  `cin.get(ch);`

  `cin` and `get` are two separate identifiers separated by a dot
- Dot separates the input stream variable name from the member or function name
- In C++, dot is the **member access operator**

`cin` and the `ignore` Function

- `ignore` discards a portion of the input
- The syntax to use the function `ignore` is:

```
cin.ignore(IntExp, CharExp);
```

  - `IntExp` is an integer expression
  - `CharExp` is a `char` expression
  - ignores the next `IntExp` characters or all characters until character `CharExp` (whichever comes first)

Statement	Input	Value Stored in Variable/Memory

`cin.ignore(3,'\n');`
`cin>>i;`
	1 2 3 4 5 New line
	6 7
		Ignores "123"
		i=45

`cin.ignore(7,'\n');`
`cin>>i;`
	1 2 3 4 5 New line
	6 7
		Ignores "12345"
		i=67

`cin` and the `putback` Function

- **`Putback`** places character back to the input stream
- The syntax for `putback`:

```
cin.putback(VarChar);
```

  - `VarChar` is a `char` variable

Statement	Input Stream	Input Stream After	Variable values

`cin.putback('9');`
`cin >> i;`
	1 2 3 New line	9 1 2 3 New line	i = 9123

`cin` and the `peek` Functions

- **`peek`** returns next character from the input stream, without removing the character from that stream
- The syntax for `peek`:

```
VarChar = cin.peek();
```

  - `VarChar` is a `char` variable

Statement	Input Stream	Input Stream After	Variable values

`c=cin.peek();`
`cin >> i;`
	1 2 3 New line	1 2 3 New line	`c = '1';`
			`i = 123`

Input Failure

- If input data does not match corresponding variables, and an error occurs when reading data, the input stream enters the fail state.
  - Trying to read a letter (character) into an `int` or `double` variable will result in an input failure
- Once in a fail state, all further I/O statements using that stream are ignored and the program continues to use whatever values are stored in variables causing incorrect results
- The **`clear`** function restores input stream to a working state (not in Visual C++)

```
cin.clear();
```

Input for the `string` Type

- An input stream variable (`cin`) and `>>` operator can read a string into a variable of the data type `string`
- Extraction operator `'>>'` skips any leading whitespace characters and reading stops at a whitespace character
  - From "`     John Doe          `" it gets only "`John`"

Input for the `string` Type

- The function `getline`
  - Reads until end of the current line

```
getline(cin, VarStr);
```

    - `VarStr` is a string variable
    - Read an entire line (until `'\n'`) from cin `istream` into the variable `VarStr`
  - Can be used for any istream for reading strings that contain non-newline whitespaces (space, tab)

Input for the `string` Type

Statement	Input	Value Stored in Variable/Memory
`cin >> s1;`	C | A | T | S | & | D | O | G | S | new line	s=“CATS&DOGS”
`cin >> s1;`	C | A | T | S | | D | O | G | S | new line	s1=“CATS” “DOGS” is held for later input

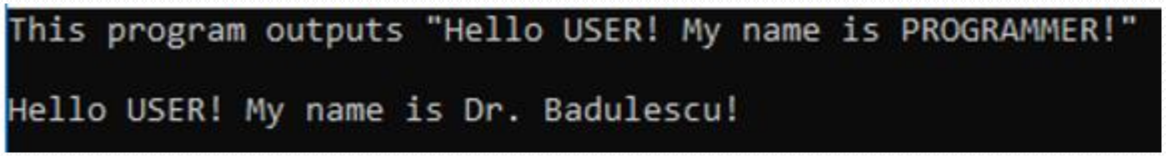
`cin >> s1`
`    >> s2;`
	C | A | T | S | | D | O | G | S | new line	s1=“CATS”
		s2=“DOGS”

`getline(cin,s);`	C | A | T | S | | D | O | G | S | new line	s=“CATS DOGS”

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!
```

Exercises with Strings

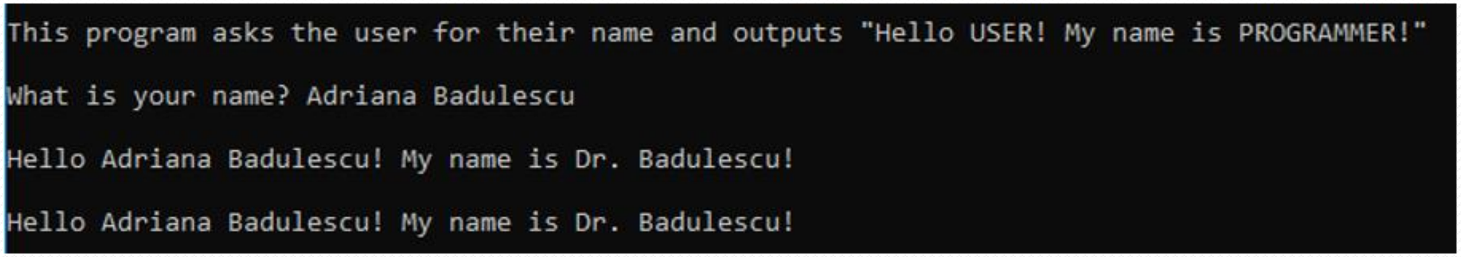
- Write a program that asks the user for their name and outputs "Hello USER! My name is PROGRAMMER!“, replace USER with the user's name and PROGRAMMER with the programmer's name

```
cout << "\n\n\nThis program asks the user for their name and "
       << "outputs \"Hello USER! My name is PROGRAMMER!\"";
//INPUT UserName
//prompt the user for their name
cout << "\n\nWhat is your name? ";
//alocate memory/declare variable
string UserName;
//read into variable
getline(cin, UserName);
```

We cannot use cin >> UserName; because the extraction operation stop at the first white space and names usually have a space (between the first and last name)

Exercises with Strings

```
//COMPUTE ProgrammerName
//declare variable
string ProgrammerName;
//assign value to variable
ProgrammerName = "Dr. Badulescu";
//OUTPUT Message  "Hello USER! My name is PROGRAMMER!"
cout << "\nHello "<<UserName<<"! My name is “
<<ProgrammerName<<"!";
//OUTPUT Message (concatenated the parts inside the output)
cout << ( "\n\nHello " + UserName + "! My name is " +
ProgrammerName + "!" ) ;
```

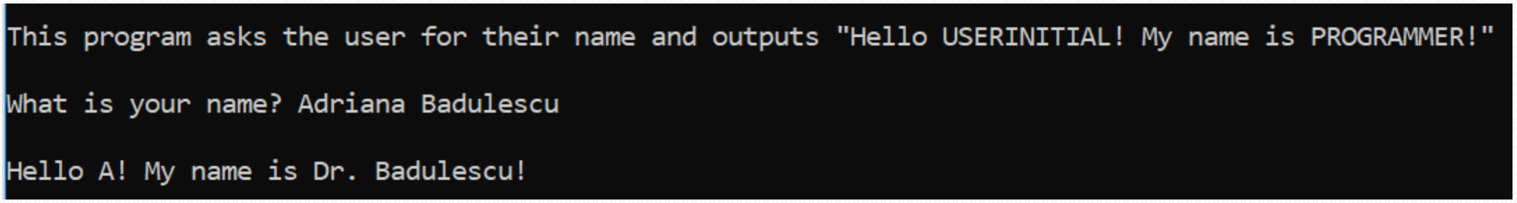


```
This program asks the user for their name and outputs "Hello USER! My name is PROGRAMMER!"
What is your name? Adriana Badulescu
Hello Adriana Badulescu! My name is Dr. Badulescu!
Hello Adriana Badulescu! My name is Dr. Badulescu!
```

Exercises with Strings

- Write a program that asks the user for their name and outputs "Hello USER! My name is PROGRAMMER!“, replace USER with the user's first name initial and PROGRAMMER with the programmer's name

```
cout << "\n\n\nThis program asks the user for their name and "
        << "outputs \"Hello USERINITIAL! My name is PROGRAMMER!\" ";
//INPUT UserName
//prompt the user for their name
cout << "\n\nWhat is your name? ";
//declare variable for initial
char Initial;
//read the initial into variable
Initial = cin.peek();
//OUTPUT Message  "Hello USER! My name is Dr. Badulescu!"
cout << "\nHello " << Initial << "! My name is Dr. Badulescu !";
```



```
This program asks the user for their name and outputs "Hello USERINITIAL! My name is PROGRAMMER!"

What is your name? Adriana Badulescu

Hello A! My name is Dr. Badulescu!
```

Output and Formatting Output

- Syntax of **`cout`** when used with **<<**

```
cout << Expression ;
cout << Manipulator ;
cout << Expression or Manipulator
     << Expression or Manipulator … ;
```

- Expression is evaluated first
- Value is printed
- **Manipulator** is used to format the output
  - Example: **`endl`**
  - `cout << "You entered the number N = " << N << endl;`

Output and Formatting 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

`setprecision` Manipulator

- Outputs decimal numbers with up to `n` decimal places
- Syntax: `setprecision(n)`
- Must include the header file **`iomanip`**

```
#include <iomanip>
```

Statement	Output

`const double   d1 = 1.23456789;`
`const double   d2 = 123.456789;`
`const double   d3 = 12345.0;`	

`cout << setprecision(1)`
`<<d1<<endl<<d2<<endl<<d3;`
	`1.2`
	`123.5`
	`12345.0`

`cout << setprecision(2)`
`<<d1<<endl<<d2<<endl<<d3;`
	`1.23`
	`123.46`
	`12345.00`

`cout << setprecision(5)`
`<<d1<<endl<<d2<<endl<<d3;`
	`1.23457`
	`123.45679`
	`12345.00000`

`fixed` and `scientific` Manipulator

- **`fixed`** outputs floating-point numbers in a fixed decimal format
  - `cout << fixed;`
  - Default flag
- **`scientific`** outputs floating-point numbers in scientific format
  - `cout << scientific;`

Statement	Output

`double d1 = 1.23456789;`
`double d2 = 123.456789;`
`cout<<setprecision(2);`	

`cout <<fixed`
`     <<d1<<endl<<d2;`
	`d1=1.23`
	`d2=123.46`

`cout <<scientific`
`     <<d1<<endl<<d2;`
	`d1=1.23e+000`
	`d2=1.23e+002`

`cout <<fixed`
`     <<d1<<endl<<d2;`
	`d1=1.23`
	`d2=123.46`

Setting and Resetting Manipulators

- Explicitly **setting** an I/O stream flag / manipulator

```
    cout << manipulator;
    cout << setiosflags(ios_base::manipulator);
```

  - `cout << scientific;`
  - `cout << setiosflags(ios_base::scientific);`

- Explicitly **resetting** an I/O stream flag / manipulator

```
    cout.unsetf(ios::manipulator);
    cout << resetiosflags(ios_base::manipulator);
```

- `cout.unsetf(ios::scientific);`
- `cout << resetiosflags(ios_base::scientific) << endl;`
- `cout << fixed;`

`showpoint` Manipulator

- **`showpoint`** forces output to show the decimal point and trailing zeros
- Examples:

Statement
`cout <<  fixed << showpoint << setprecision(1) << "\t" << static_cast<double>(1) << "\t" << .6;`

`setw` Manipulator

`#include <iomanip>`

- Outputs the value of an expression in specific columns
- Output of the expression is right-justified
- Unused columns to the left are filled with spaces

`setw(n)`

Statement	Output

`double d1 = 1.23456789;`
`double d2 = 123.456789;`
`double d3 = 1234567;`
`cout<<setprecision(1);`	

`cout<<setw(3)<<d1<<endl;`
`cout<<setw(3)<<d2<<endl;`
`cout<<setw(3)<<d3<<endl;`
	`1.2`
	`123.5`
	`1234567.0`

`cout<<setw(10)<<d1<<endl;`
`cout<<setw(10)<<d2<<endl;`
`cout<<setw(10)<<d3<<endl;`
	`       1.2`
	`     123.5`
	` 1234567.0`

setfill Manipulator

```
#include <iomanip>
```

- Output stream variables can use **setfill** to fill unused columns with a character

```
setfill(ch)
```

Statement	Output

`double d1 = 1.23456789;`
`double d2 = 123.456789;`
`double d3 = 1234567;`
`cout<<setprecision(1);`
`cout << setfill('_');`	

`cout<<setw(10)<<d1<<endl;`
`cout<<setw(10)<<d2<<endl;`
`cout<<setw(10)<<d3<<endl;`
	`_______1.2`
	`_____123.5`
	`_1234567.0`

left and right Manipulators

- **left**: left-justifies the output
- **right**: right-justifies the output

```
cout << left;
```

```
cout << right;
```

Statement	Output

`double d1 = 1.23456789;`
`double d2 = 123.456789;`
`double d3 = 1234567;`
`cout<<setprecision(1);`	

`cout<<left;`
`cout<<setw(10)<<d1<<endl;`
`cout<<setw(10)<<d2<<endl;`
`cout<<setw(10)<<d3<<endl;`
	`1.2       `
	`123.5     `
	`1234567.0 `

`cout<<right;`
`cout<<setw(10)<<d1<<endl;`
`cout<<setw(10)<<d2<<endl;`
`cout<<setw(10)<<d3<<endl;`
	`       1.2`
	`     123.5`
	` 1234567.0`

Types of Manipulators

- Two **types of manipulators**:
  - **Parameterized** - with parameters
    - require `iomanip` and `iostream` headerS
    - `setprecision`, `setw`, and `setfill`
  - **Nonparameterized** - without parameters
    - require `iostream` header
    - `endl`, `fixed`, `scientific`, `showpoint` ,`left`, and `right`

File Input/Output

- **File**: area in secondary storage to hold information
- File I/O is a five-step process
  1. Include **`fstream`** header

     ```
     #include <fstream>
     ```

     ```
     #include <fstream>
     ```

  2. Declare file stream variables

     ```
     ifstream FilestreamVariable;

     ofstream FilestreamVariable;
     ```

File Input/Output

3. Associate the file stream variables with the input/output sources (opening the file)
   - `FilestreamVariable.open(SourceFileName, Mode);`
     - `FilestreamVariable` is a file stream variable (in or out)
     - `SourceFileName` is the name of the input/output file with optional path
     - `Mode` is the opening mode like `ios::in`, `ios::out`, `ios::nocreate`, `ios::app`

```
in.open("InputFile.txt",ios::in);
out.open("OutputFile.txt",ios::out|ios::app);
```

   - You have to add these source file to the project and need to use the absolute path if the file is in the folder that has the CPP file
   - If the file does not exist, the system creates an empty file

File Input/Output

- If you use the default input and output modes (`ios::in`, `ios::out`), you can do step 2 and 3 in one step

```
ifstream FilestreamVariable(SourceFileName);
ofstream FilestreamVariable(SourceFileName);
```

Declare and Open	Open at Declaration

`ifstream in;`
`in.open("InputFile.txt",ios::in);`
	`ifstream in ("InputFile.txt");`

`ofstream out;`
`out.open("OutputFile.txt",ios::out);`
	`ofstream out("OutputFile.txt");`

File Input/Output

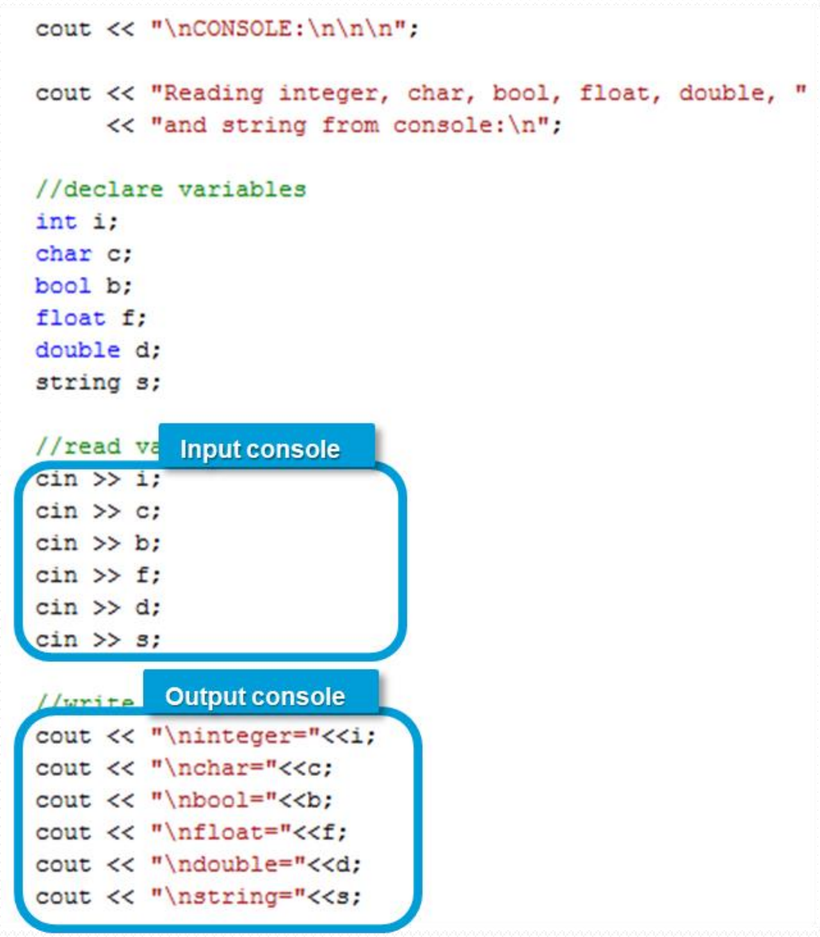
4. Use the file stream variables with >>, <<, or other input/output functions

```
FilestreamVariable >> variable;
FilestreamVariable << expression or
   manipulator;
```

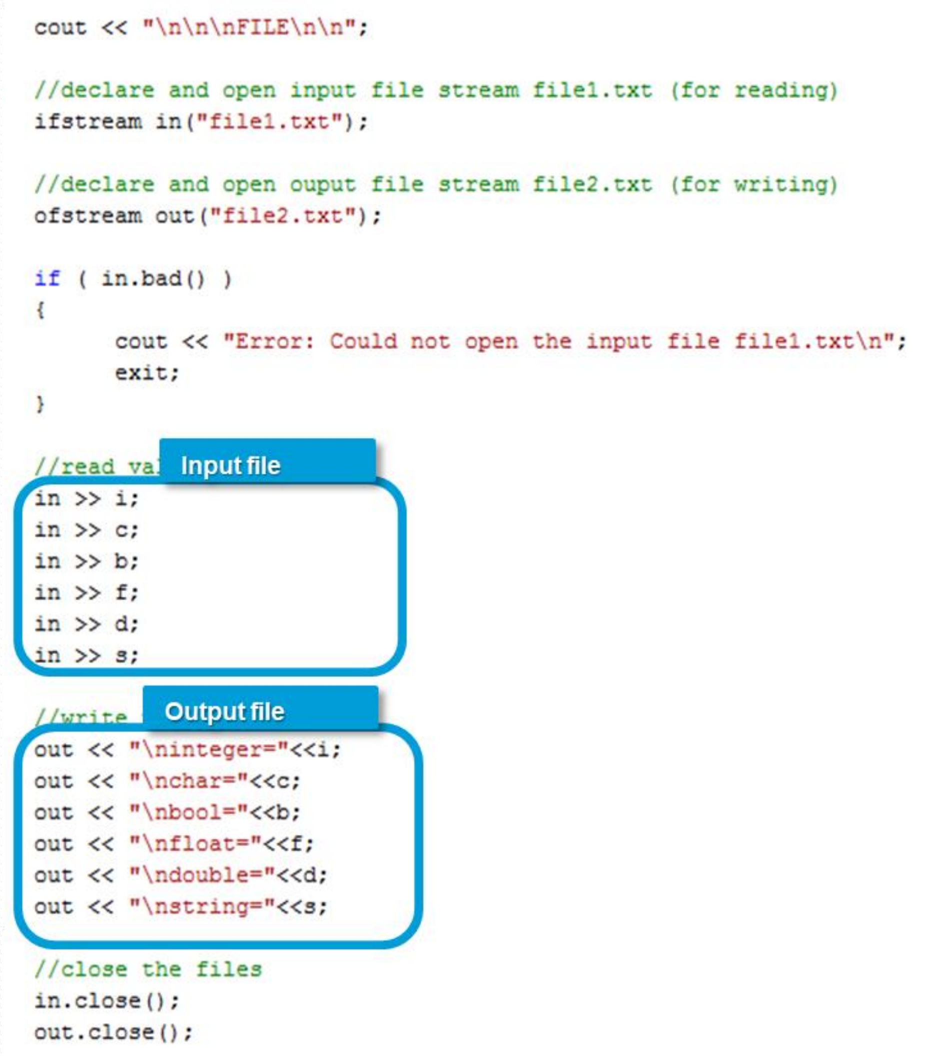
5. Close the files

```
FilestreamVariable.close();
```

File Input/Output

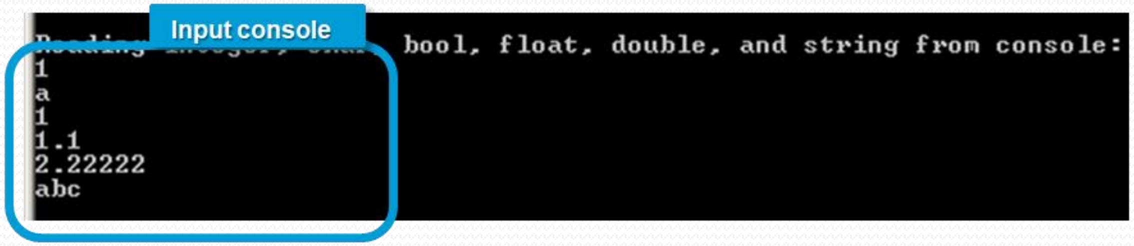


```
cout << "\nCONSOLE:\n\n\n";

cout << "Reading integer, char, bool, float, double, "
     << "and string from console:\n";

//declare variables
int i;
char c;
bool b;
float f;
double d;
string s;

//read va
cin >> i;
cin >> c;
cin >> b;
cin >> f;
cin >> d;
cin >> s;

//write
cout << "\ninteger="<<i;
cout << "\nchar="<<c;
cout << "\nbool="<<b;
cout << "\nfloat="<<f;
cout << "\ndouble="<<d;
cout << "\nstring="<<s;
```

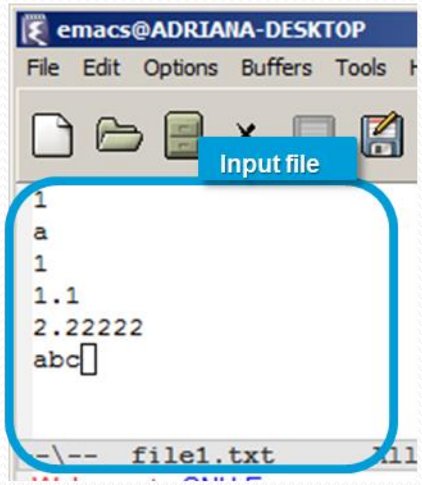


```
cout << "\n\n\nFILE\n\n";

//declare and open input file stream file1.txt (for reading)
ifstream in("file1.txt");

//declare and open ouput file stream file2.txt (for writing)
ofstream out("file2.txt");

if ( in.bad() )
{
      cout << "Error: Could not open the input file file1.txt\n";
      exit;
}

//read va
in >> i;
in >> c;
in >> b;
in >> f;
in >> d;
in >> s;

//write
out << "\ninteger="<<i;
out << "\nchar="<<c;
out << "\nbool="<<b;
out << "\nfloat="<<f;
out << "\ndouble="<<d;
out << "\nstring="<<s;

//close the files
in.close();
out.close();
```

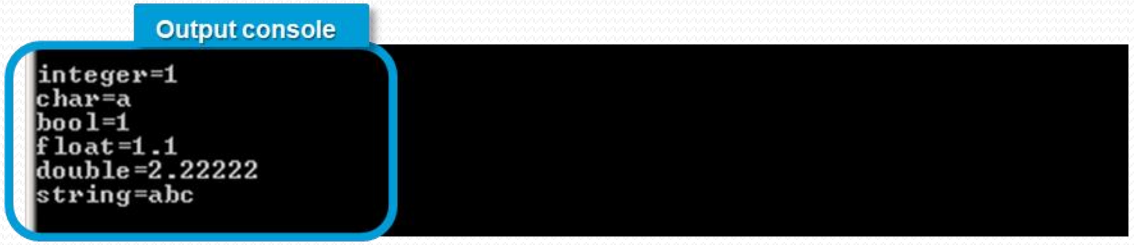
File Input/Output



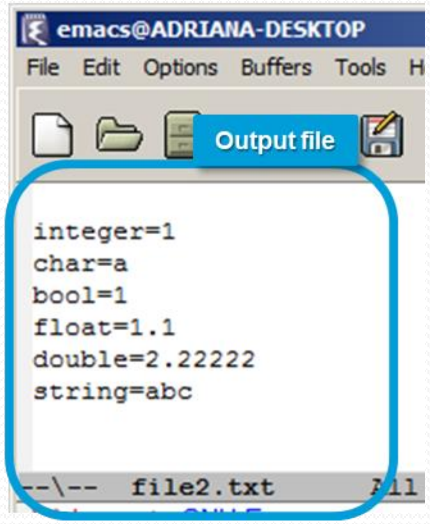
```
Reading integer, char, bool, float, double, and string from console:
1
a
1
1.1
2.22222
abc
```



```
1
a
1
1.1
2.22222
abc
```



```
integer=1
char=a
bool=1
float=1.1
double=2.22222
string=abc
```



```
integer=1
char=a
bool=1
float=1.1
double=2.22222
string=abc
```

Exercises With Files

- Write a program that reads data from a formatted text file that have on each line a number and a name and output it in a formatted table: with columns, header/heading, and lines/borders. The Input.txt file content is this:

```
1234      Alexander
-2.3456 Brenda
3.456789       Candy
```

Exercises With Files

- First, you need the InputText.txt file , so, you should either
  - create a Text file using the File Explorer in the same folder as the CPP file for the project (if you place it anywhere else, you will need to specify the exact path to the file and you will not be able to run the code in any other computer) and then add it a an Existing Resource to the project or
  - create a New Resource – Text File and save it in the same folder as the CPP file
- Then, add the following content to the file (separated by tabs and new lines):

```
 1234       Alexander
-2.3456     Brenda
3.456789    Candy
```

Exercises With Files

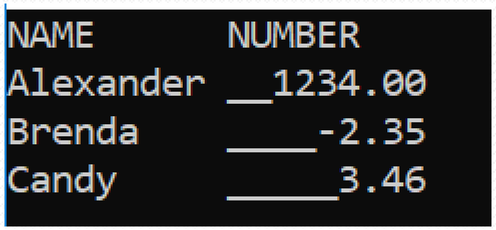
- Read from file

```
//declare and open the file
ifstream in("InputFile.txt");
//read from the file into variables
//read Number1
float Number1;
in >> Number1;
//read Name1
string Name1;
in >> Name1;
//read Number2
float Number2;
in >> Number2;
//read Name2
string Name2;
in >> Name2;
//read Number3
float Number3;
in >> Number3;
//read Name3
string Name3;
in >> Name3;
//close the file
in.close();
```

Exercises With Files

- Output the data into a table format

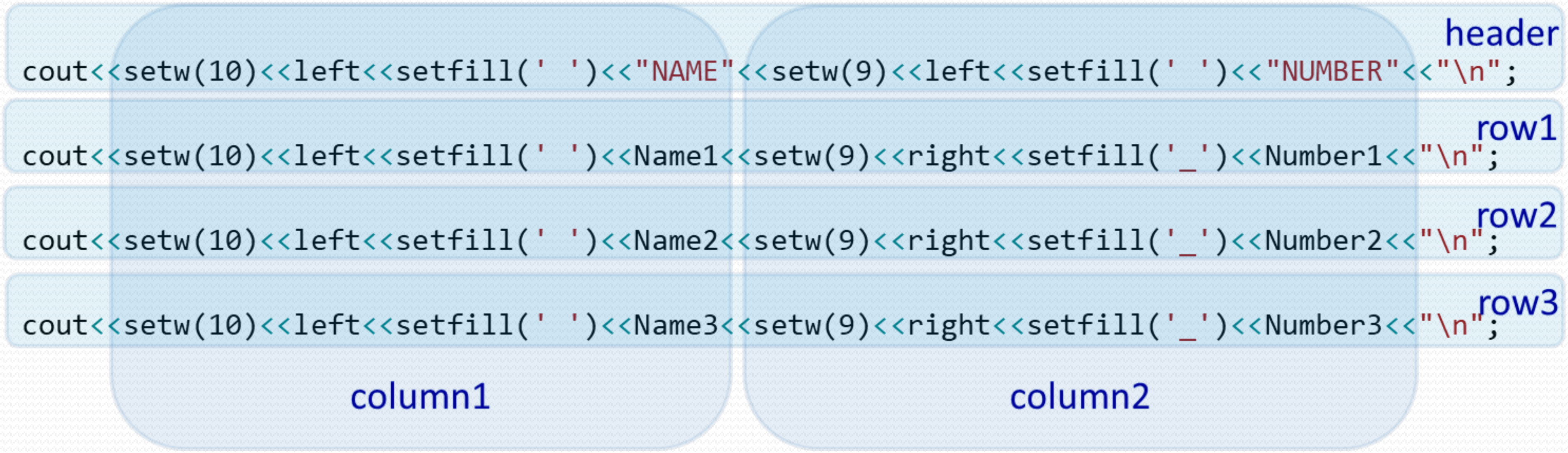
```
//output the data from the variables
//format the floating points numbers
cout << fixed << showpoint << setprecision(2);
//header
cout << setw(10) << left << setfill(' ') << "NAME"
         << setw(9) << left << setfill(' ') << "NUMBER"
         << "\n";
//row1
cout << setw(10) << left << setfill(' ') << Name1
         << setw(9) << right << setfill('_') << Number1
         << "\n";
//row2
cout << setw(10) << left << setfill(' ') << Name2
         << setw(9) << right << setfill('_') << Number2
         << "\n";
//row3
cout << setw(10) << left << setfill(' ') << Name3
         << setw(9) << right << setfill('_') << Number3
         << "\n";
```



```
NAME      NUMBER
Alexander __1234.00
Brenda    ____-2.35
Candy     _____3.46
```

Exercises With Files

- If we break the output this where each row and column goes



```
cout<<setw(10)<<left<<setfill(' ')<<"NAME"<<setw(9)<<left<<setfill(' ')<<"NUMBER"<<"\n";
cout<<setw(10)<<left<<setfill(' ')<<Name1<<setw(9)<<right<<setfill('_')<<Number1<<"\n";
cout<<setw(10)<<left<<setfill(' ')<<Name2<<setw(9)<<right<<setfill('_')<<Number2<<"\n";
cout<<setw(10)<<left<<setfill(' ')<<Name3<<setw(9)<<right<<setfill('_')<<Number3<<"\n";
```

Exercises With Files

- Output the data into a table format (with ASCII lines)

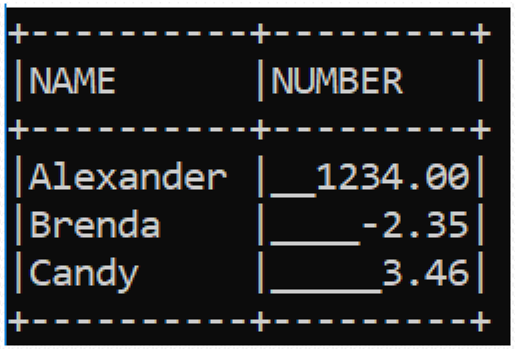
```
//output the data from the variables
//format the floating points numbers
cout << fixed << showpoint << setprecision(2);
//horizontal line
cout << "+"
         << setw(10) << left << setfill('-') << "-"
         << "+"
         << setw(9) << left << setfill('-') << "-"
         << "+"
         << "\n";
//header
cout << "|"
         << setw(10) << left << setfill(' ') << "NAME"
         << "|"
         << setw(9) << left << setfill(' ') << "NUMBER"
         << "|"
         << "\n";
```

Exercises With Files

```
//horizontal line
cout << "+"
         << setw(10) << left << setfill('-') << "-"
         << "+"
         << setw(9) << left << setfill('-') << "-"
         << "+"
         << "\n";
//row1
cout << "|"
         << setw(10) << left << setfill(' ') << Name1
         << "|"
         << setw(9) << right << setfill('_') << Number1
         << "|"
         << "\n";
//row2
cout << "|"
         << setw(10) << left << setfill(' ') << Name2
         << "|"
         << setw(9) << right << setfill('_') << Number2
         << "|"
         << "\n";
```

Exercises With Files

```
//row3
cout << "|"
         << setw(10) << left << setfill(' ') << Name3
         << "|"
         << setw(9) << right << setfill('_') << Number3
         << "|"
         << "\n";
//horizontal line
cout << "+"
         << setw(10) << left << setfill('-') << "-"
         << "+"
         << setw(9) << left << setfill('-') << "-"
         << "+"
         << "\n";
```

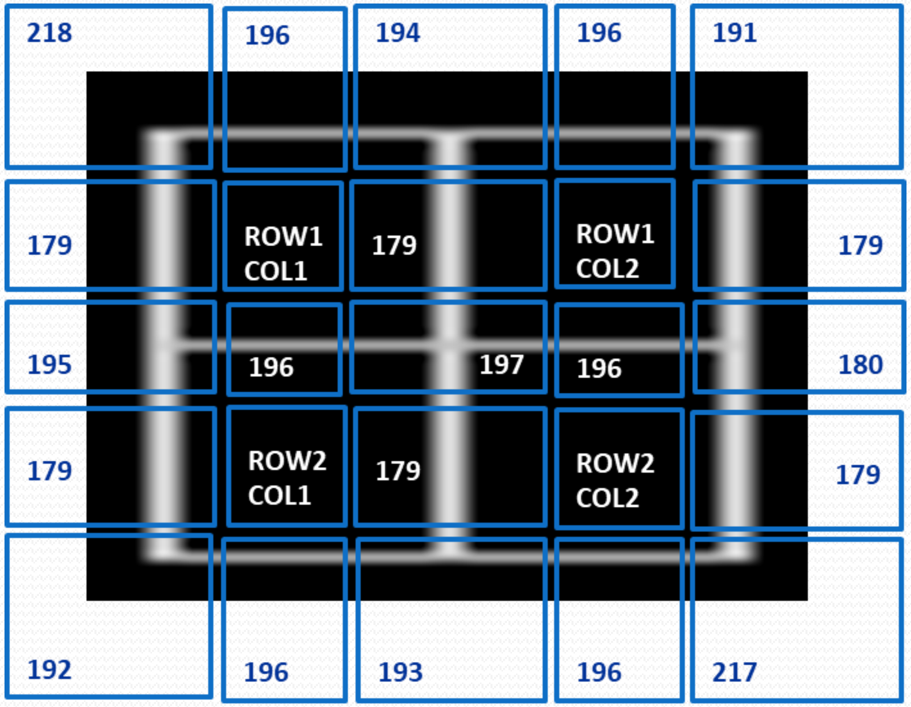


```
+----------+---------+
|NAME      |NUMBER   |
+----------+---------+
|Alexander |__1234.00|
|Brenda    |____-2.35|
|Candy     |_____3.46|
+----------+---------+
```

Using Extended ASCII characters

- To output Extended ASCII characters, you need to output the extended ASCII code as a character (cast it to a char)

```
cout << static_cast<char>(218)
```



Exercises With Files

- Output the data into a table format, with Extended ASCII lines

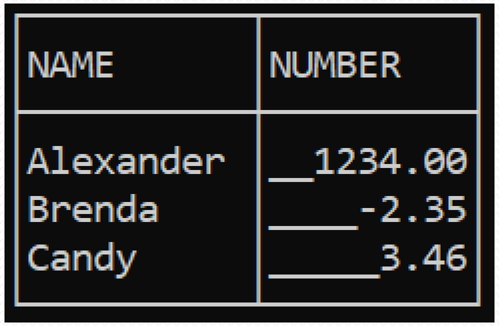
```
//top horizontal line
cout << static_cast<char>(218)
         << setw(10) << left << setfill(static_cast<char>(196)) << static_cast<char>(196)
         << static_cast<char>(194)
         << setw(9) << left << setfill(static_cast<char>(196)) << static_cast<char>(196)
         << static_cast<char>(191)
         << "\n";
//header
cout << static_cast<char>(179)
         << setw(10) << left << setfill(' ') << "NAME"
         << static_cast<char>(179)
         << setw(9) << left << setfill(' ') << "NUMBER"
         << static_cast<char>(179)
         << "\n";
//horizontal line
cout << static_cast<char>(195)
         << setw(10) << left << setfill(static_cast<char>(196)) << static_cast<char>(196)
         << static_cast<char>(197)
         << setw(9) << left << setfill(static_cast<char>(196)) << static_cast<char>(196)
         << static_cast<char>(180)
         << "\n";
```

Exercises With Files

```
//row1
cout << static_cast<char>(179)
           << setw(10) << left << setfill(' ') << Name1
           << static_cast<char>(179)
           << setw(9) << right << setfill('_') << Number1
           << static_cast<char>(179)
           << "\n";
//row2
cout << static_cast<char>(179)
           << setw(10) << left << setfill(' ') << Name2
           << static_cast<char>(179)
           << setw(9) << right << setfill('_') << Number2
           << static_cast<char>(179)
           << "\n";
//row3
cout << static_cast<char>(179)
           << setw(10) << left << setfill(' ') << Name3
           << static_cast<char>(179)
           << setw(9) << right << setfill('_') << Number3
           << static_cast<char>(179)
           << "\n";
```

Exercises With Files

```
//bottom horizontal line
cout << static_cast<char>(192)
         << setw(10) << left << setfill(static_cast<char>(196)) << static_cast<char>(196)
         << static_cast<char>(193)
         << setw(9) << left << setfill(static_cast<char>(196)) << static_cast<char>(196)
         << static_cast<char>(217)
         << "\n";
```



```
┌──────────┬─────────┐
│NAME      │NUMBER   │
├──────────┼─────────┤
│Alexander │__1234.00│
│Brenda    │____-2.35│
│Candy     │_____3.46│
└──────────┴─────────┘
```

Summary

- Input and output streams
- Input operators and methods/functions
- Output operators and manipulators
- Input and output from files