Programming Techniques I SCJ1013

Input & Output Operations

Dr Masitah Ghazali

Formatting Output

Formatting Output

- Can control how output displays for numeric, string data:
  - size
  - position
  - number of digits
- Requires **`iomanip`** header file

Formatting Output

- Used to control how an output field is displayed

- Some affect just the next value displayed:
  - **`setw(x)`** : print in a field at least **`x`** spaces wide. Use more spaces if field is not wide enough

Formatting Output – setw(*n*)

- Default setw is to the right
- Can be written as:

```
cout<< left;
cout<< setw(10) <<n;
```

OR

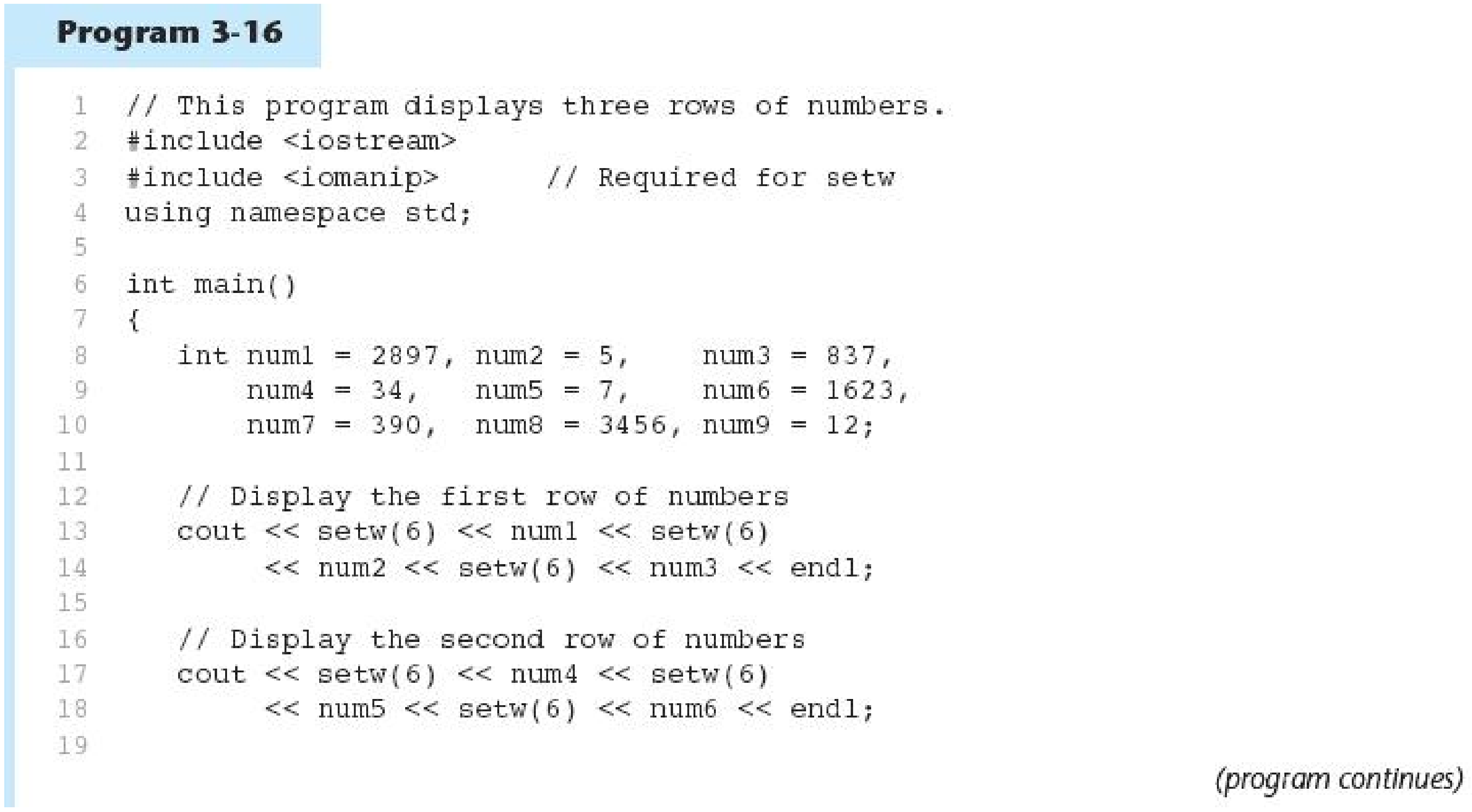
```
cout<< setw(-10) <<n;
```

- Example

```
cout<< "Enter an integer:";
cin>>n;


cout<<n<<endl;
cout<< setw(6) <<n<<endl;
cout<< setw(-6) <<n<<endl;
```

```
Enter an integer: 5
5
-----5
5
```

Formatting Output - example

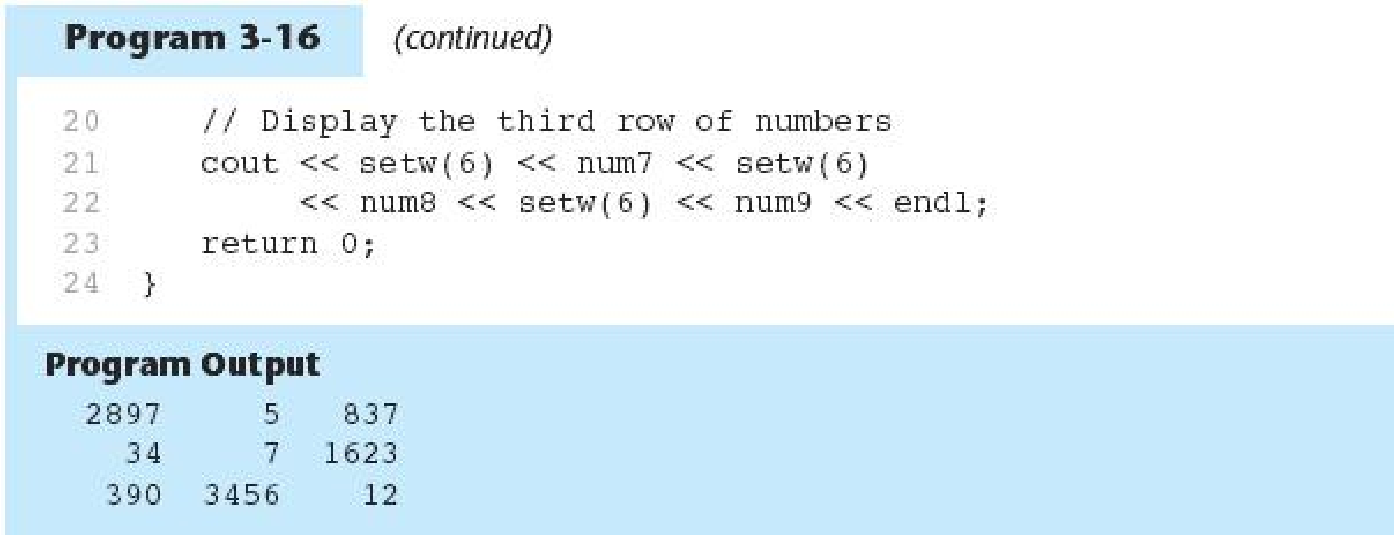


Program 3-16

```
// This program displays three rows of numbers.
#include <iostream>
#include <iomanip>      // Required for setw
using namespace std;

int main()
{
   int num1 = 2897, num2 = 5,    num3 = 837,
       num4 = 34,   num5 = 7,    num6 = 1623,
       num7 = 390,  num8 = 3456, num9 = 12;

   // Display the first row of numbers
   cout << setw(6) << num1 << setw(6)
        << num2 << setw(6) << num3 << endl;

   // Display the second row of numbers
   cout << setw(6) << num4 << setw(6)
        << num5 << setw(6) << num6 << endl;

```

(program continues)

Formatting Output - example



Program 3-16 *(continued)*

```
    // Display the third row of numbers
    cout << setw(6) << num7 << setw(6)
         << num8 << setw(6) << num9 << endl;
    return 0;
}
```

Program Output

```
  2897     5   837
    34     7  1623
   390  3456    12
```

Stream Manipulators

- Some affect values until changed again:
  - **`fixed`**: use decimal notation for floating-point values
  - **`setprecision(x)`** : when used with **`fixed`**, print floating-point value using **`x`** digits after the decimal. Without **`fixed`**, print floating-point value using x significant digits
  - **`showpoint`**: always print decimal for floating-point values

Formatting Output – fixed

- Always print out 6 digits after the decimal notation

```
cout << "input one floating number: ";
cin >> f;
cout << fixed << f << endl;
```

```
Enter a floating number: 3.1
3.100000
```

```
Enter one double number: 1234.567
1234.567000
```

```
Enter a floating number: 3.4565679
3.456568
```

```
Enter one double number: 1234567.4
1234567.400000
```

Formatting Output – setprecision(x)

- When used without **fixed**, print floating-point value using **x**significant digits

```
cout << "enter one double number: ";
cin >> d;
cout << d << endl;
cout << setprecision(5) << d << endl;
```

```
Enter one double number: 3.1
3.1
```

```
Enter one double number: 1234.567
1234.6
```

```
Enter one double number: 1234567.4
1.2346e+006
```

Formatting Output – setprecision(x) with fixed

- when used with **`fixed`**, print floating-point value using **`x`** digits after the decimal.

```
cout << "enter one double number: ";
cin >> d;
cout << d << endl;
cout << fixed << setprecision(3) << d << endl;
```

```
Enter one double number: 3.1
3.100
```

```
Enter one double number: 1234.567
1234.567
```

```
Enter one double number: 1234567.4
1234567.400
```

Formatting Output – showpoint

- always print decimal for floating-point values

```
cout << "input one floating number: ";
cin >> f;
cout << showpoint << f << endl;
```

```
Enter a floating number: 3.1
3.10000
```

```
Enter one double number: 1234.567
1234.57
```

```
Enter a floating number: 3.4565679
3.45657
```

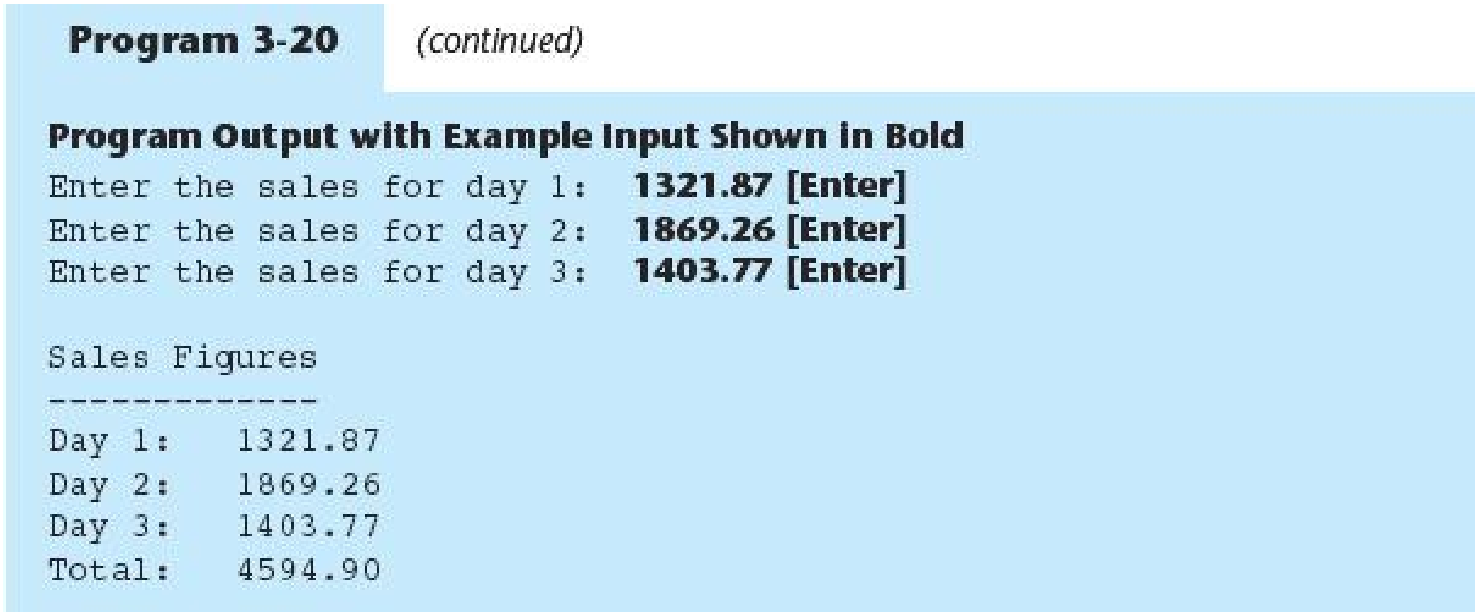
```
Enter one double number: 1234567.4
1.23457e+006
```

Stream Manipulators – example

Program 3-20

```
// This program asks for sales figures for 3 days. The total
// sales are calculated and displayed in a table.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   double day1, day2, day3, total;

   // Get the sales for each day.
   cout << "Enter the sales for day 1: ";
   cin >> day1;
   cout << "Enter the sales for day 2: ";
   cin >> day2;
   cout << "Enter the sales for day 3: ";
   cin >> day3;

   // Calculate the total sales.
   total = day1 + day2 + day3;

   // Display the sales figures.
   cout << "\nSales Figures\n";
   cout << "-------------\n";
   cout << setprecision(2) << fixed;
   cout << "Day 1: " << setw(8) << day1 << endl;
   cout << "Day 2: " << setw(8) << day2 << endl;
   cout << "Day 3: " << setw(8) << day3 << endl;
   cout << "Total: " << setw(8) << total << endl;
   return 0;
}
```

Stream Manipulators – example



Program 3-20 *(continued)*

Program Output with Example Input Shown in Bold

```
Enter the sales for day 1:  1321.87 [Enter]
Enter the sales for day 2:  1869.26 [Enter]
Enter the sales for day 3:  1403.77 [Enter]

Sales Figures
-------------
Day 1:    1321.87
Day 2:    1869.26
Day 3:    1403.77
Total:    4594.90
```

Stream Manipulators

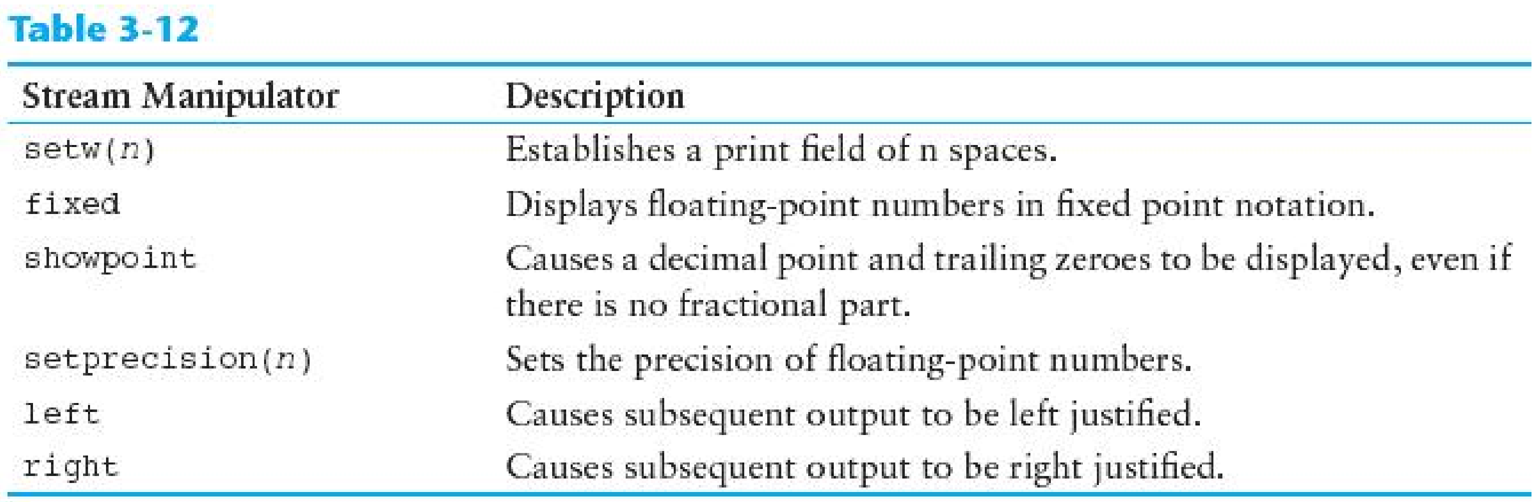


Table 3-12

Stream Manipulator	Description
`setw(n)`	Establishes a print field of n spaces.
`fixed`	Displays floating-point numbers in fixed point notation.
`showpoint`	Causes a decimal point and trailing zeroes to be displayed, even if there is no fractional part.
`setprecision(n)`	Sets the precision of floating-point numbers.
`left`	Causes subsequent output to be left justified.
`right`	Causes subsequent output to be right justified.

Exercise Week 6_1

- Refer to Exercise 2 No. 3 in pg. 76.
- Solve the problem

- Refer back to Exercise 3 No. 3 in pg. 80.
- Solve the problem by setting the output to 2 decimal places.

Formatted Input

Formatted Input

- Can format field width for use with **`cin`**
- Useful when reading string data to be stored in a character array:

```
const int SIZE = 10;
 char firstName[SIZE];
   cout << "Enter your name: ";
   cin >> setw(SIZE) >> firstName;
```

- **`cin`** reads one less character than specified with the **`setw()`** manipulator

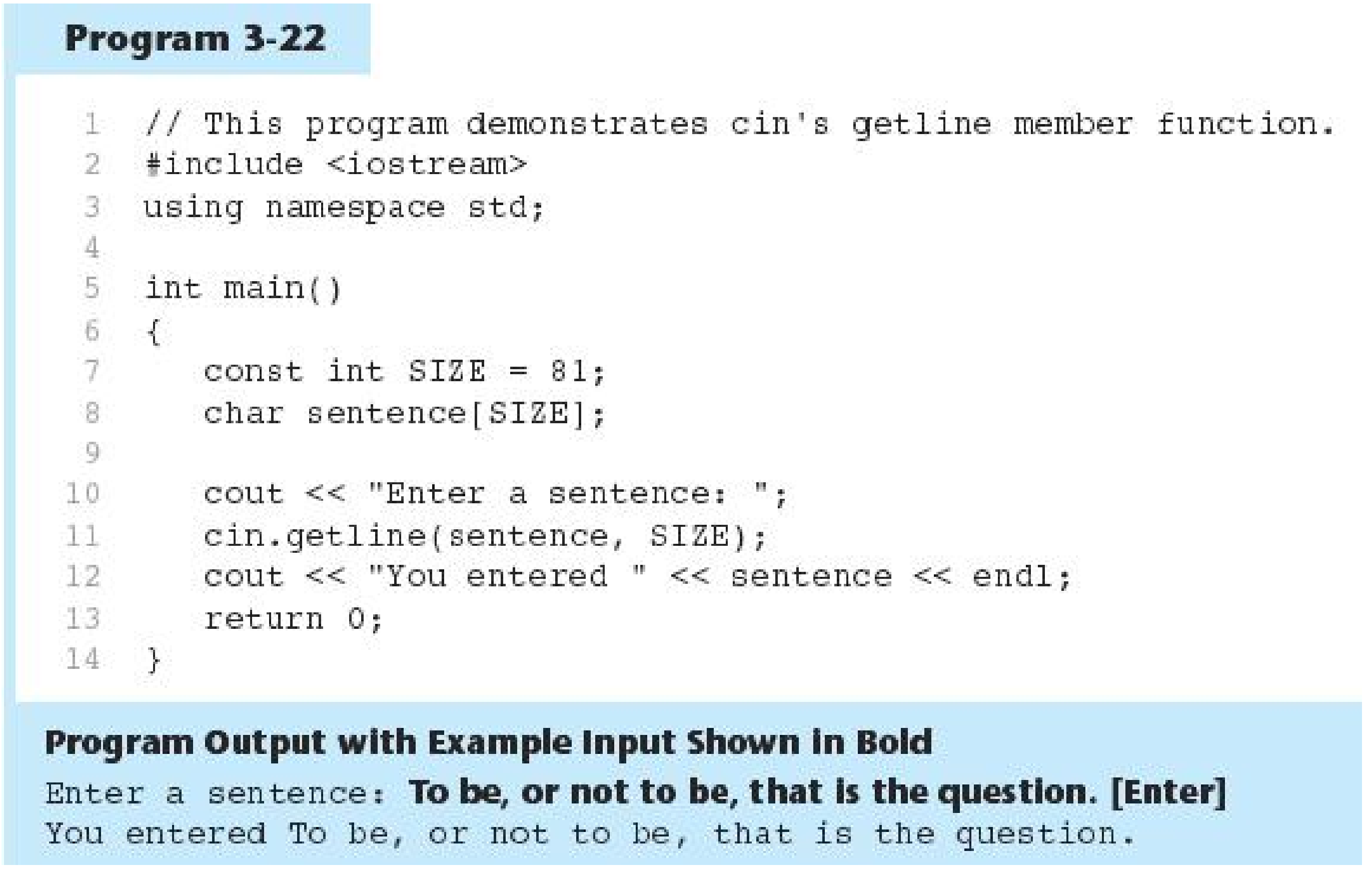
Formatted Input

- To read an entire line of input, use
  `cin.getline()`:

```
const int SIZE = 81;
  char address[SIZE];
  cout << "Enter your address: ";
  cin.getline(address, SIZE);
```

- **`cin.getline`** takes two arguments:
  - Name of array to store string
  - Size of the array

Formatted Input



Program 3-22

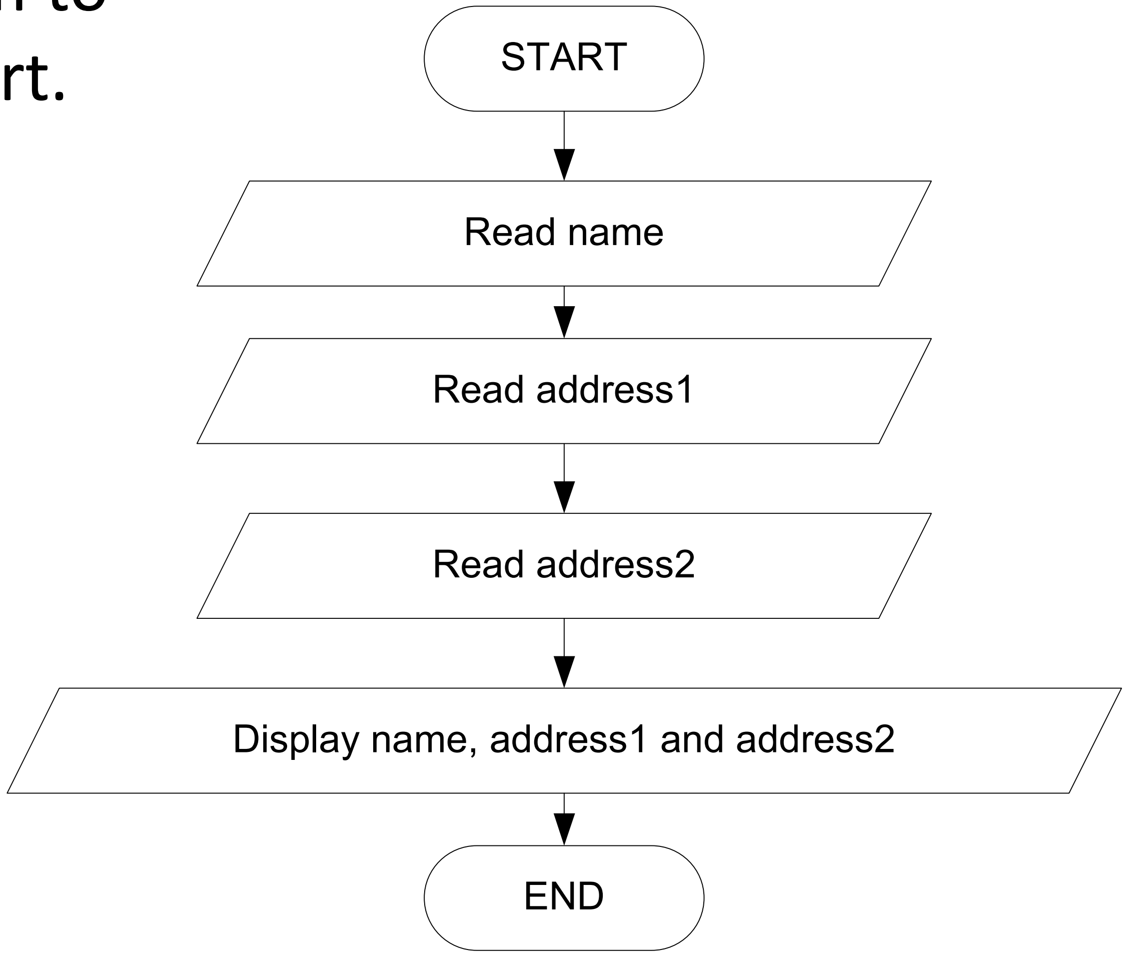
```
// This program demonstrates cin's getline member function.
#include <iostream>
using namespace std;

int main()
{
   const int SIZE = 81;
   char sentence[SIZE];

   cout << "Enter a sentence: ";
   cin.getline(sentence, SIZE);
   cout << "You entered " << sentence << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter a sentence: To be, or not to be, that is the question. [Enter]
You entered To be, or not to be, that is the question.
```

Exercise Week 6_2

- Write C++ program to solve the flow chart.



Formatted Input

- To read a single character:
  - Use `cin`:

    ```
    char ch;
    cout << "Strike any key to continue";
    cin >> ch;
    ```

    Problem: will skip over blanks, tabs, `<CR>`
  - Use `cin.get()`:

    ```
    cin.get(ch);
    ```

    Will read the next character entered, even whitespace

Exercise Week 6_3

- Refer to Exercise 2 No. 1 in pg. 74.

- What will be displayed if the following characters are entered in Program 6.2 & 6.3? Explain the program output with the following input.

  AV

  TY

Formatted Input

- Mixing **`cin >>`** and **`cin.get()`** in the same program can cause input errors that are hard to detect
- To skip over unneeded characters that are still in the keyboard buffer, use **`cin.ignore()`**:

```
 cin.ignore(); // skip next char
   cin.ignore(10, '\n'); // skip the next
 // 10 char. or until a '\n'
```

Hand Tracing a Program

Hand Tracing a Program

- Hand trace a program: act as if you are the computer, executing a program:
  - step through and ‘execute’ each statement, one-by-one
  - record the contents of variables after statement execution, using a hand trace chart (table)
- Useful to locate logic or mathematical errors

Hand Tracing a Program

Program 3-26 **(with hand trace chart filled)**

```
// This program asks for three numbers, then
// displays the average of the numbers.
#include <iostream>
using namespace std;

int main()

{

   double num1, num2, num3, avg;

   cout << "Enter the first number: ";

   cin >> num1;

   cout << "Enter the second number: ";

   cin >> num2;

   cout << "Enter the third number: ";

   cin >> num3;

   avg = num1 + num2 + num3 / 3;

   cout << "The average is " << avg << endl;

   return 0;

}
```

	num1	num2	num3	avg
`double num1, num2, num3, avg;`	?	?	?	?
`cout << "Enter the first number: ";`	?	?	?	?
`cin >> num1;`	10	?	?	?
`cout << "Enter the second number: ";`	10	?	?	?
`cin >> num2;`	10	20	?	?
`cout << "Enter the third number: ";`	10	20	?	?
`cin >> num3;`	10	20	30	?
`avg = num1 + num2 + num3 / 3;`	10	20	30	40
`cout << "The average is " << avg << endl;`	10	20	30	40

Exercise Week 6_4

- Trace the following programs

```
void main(){ //Prog 6_41
    int x, y, z;

    x =10; y = 17;
    z = x + y;
    y = y - x;
    cout<<"x: "<<x<< " y: "
<<y<<" z: "<<z;
    x = y * z;
    z = x / 20;
    y = z % x;
  cout<<"\nx: "<<x<< " y: "
<<y<<" z: "<<z;
     getch();
}
```

```
void main(){//Prog 6_42
      int n, m, x, y;
      m=10;
      n=m*2/(m+2);
      m%=n+2;
      cout <<"n: "<<n;
      cout <<"\nm: "<<m;

      x=4;
      y=x*2+10%3-1*x;
      x*=(y/m);
      cout<<"\nx:  "<< x;
      cout<<"\ny: "<<y;
      getch();
}
```

Introduction to File Input and Output

Introduction to File Input and Output

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
  - *Open* the file
  - *Use* the file (read from, write to, or both)
  - *Close* the file

Files: What is Needed

- Use `fstream` header file for file access
- File stream types:

  `ifstream` for input from a file

  `ofstream` for output to a file

  `fstream` for input from or output to a file
- Define file stream objects:

```
ifstream infile;
  ofstream outfile;
```

Opening Files

- Create a link between file name (outside the program) and file stream object (inside the program)
- Use the `open` member function:

```
infile.open("inventory.dat");
outfile.open("report.txt");
```

- Filename may include drive, path info.
- Output file will be created if necessary; existing file will be erased first
- Input file must exist for `open` to work

Using Files

- Can use output file object and << to send data to a file:

```
outfile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables:

```
infile >> partNum;
  infile >> qtyInStock >> qtyOnOrder;
```

Closing Files

- Use the `close` member function:

```
 infile.close();
  outfile.close();
```

- Don’t wait for operating system to close files at program end:
  - may be limit on number of open files
  - may be buffered output data waiting to send to file

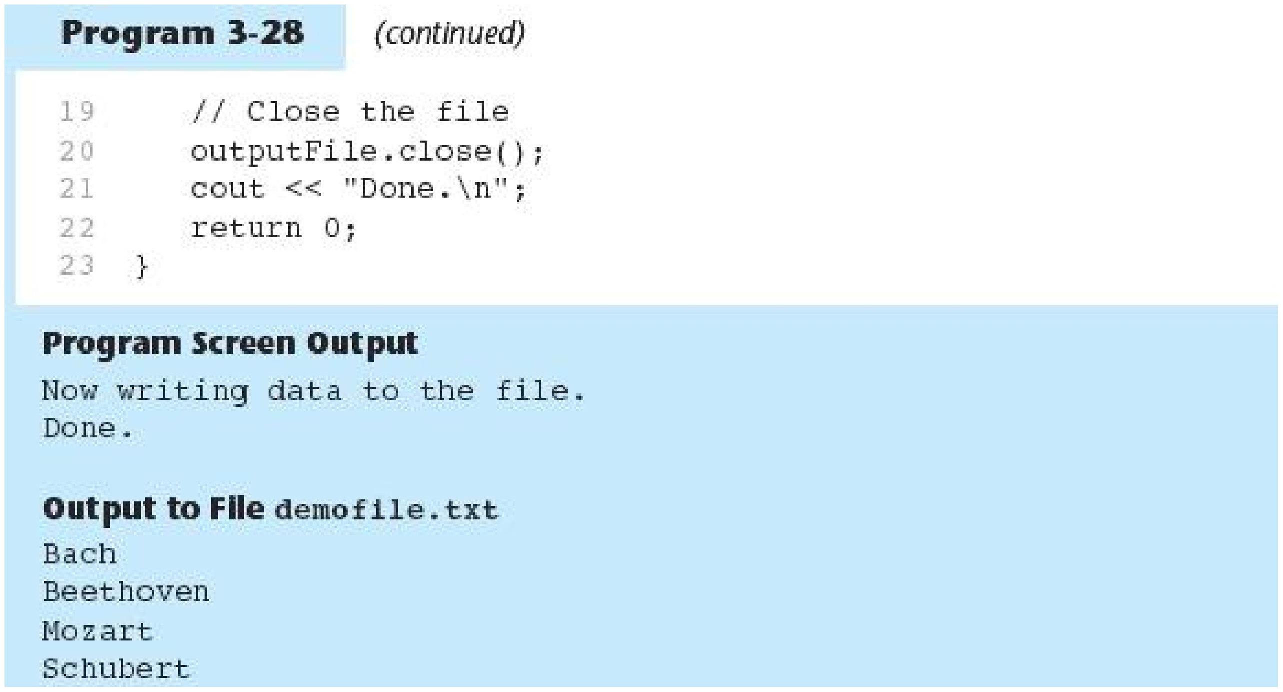
Closing Files - example

Program 3-28

```
// This program writes data to a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ofstream outputFile;
   outputFile.open("demofile.txt");

   cout << "Now writing information to the file.\n";

   // Write 4 great names to the file
   outputFile << "Bach\n";
   outputFile << "Beethoven\n";
   outputFile << "Mozart\n";
   outputFile << "Schubert\n";

```

(program continues)

Closing Files - example



Program 3-28 *(continued)*

```
    // Close the file
    outputFile.close();
    cout << "Done.\n";
    return 0;
}
```

Program Screen Output

```
Now writing data to the file.
Done.
```

Output to File `demofile.txt`

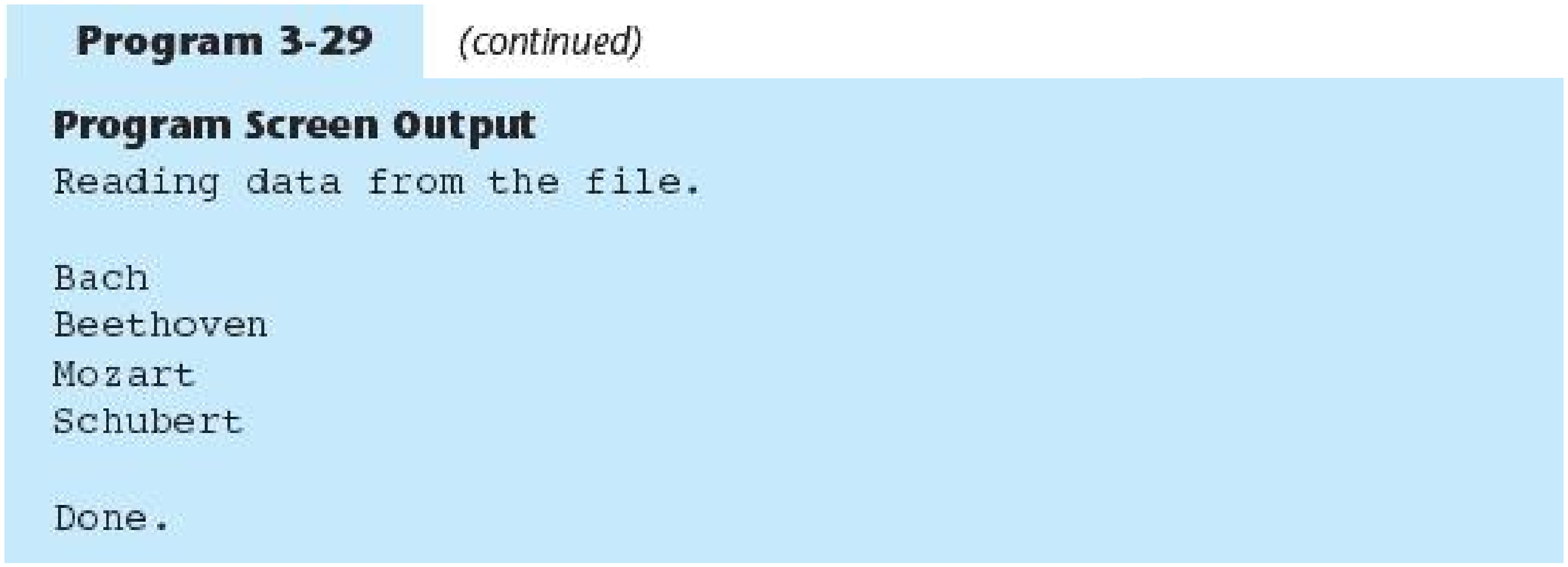
```
Bach
Beethoven
Mozart
Schubert
```

Closing Files - example

Program 3-29

```
// This program reads information from a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ifstream inFile;
   const int SIZE = 81;
   char name[SIZE];

   inFile.open("demofile.txt");
   cout << "Reading information from the file.\n\n";

   inFile >> name;          // Read name 1 from the file
   cout << name << endl;    // Display name 1

   inFile >> name;          // Read name 2 from the file
   cout << name << endl;    // Display name 2

   inFile >> name;          // Read name 3 from the file
   cout << name << endl;    // Display name 3

   inFile >> name;          // Read name 4 from the file
   cout << name << endl;    // Display name 4

   inFile.close();          // Close the file
   cout << "\nDone.\n";
   return 0;
}
```

Closing Files - example



Program 3-29 *(continued)*

Program Screen Output

```
Reading data from the file.

Bach
Beethoven
Mozart
Schubert

Done.
```

Exercise Week 6_5

- Refer to Exercise 2 No. 2 (i-iv) in pg. 75-76.
- Solve the problem

Thank You

Q & A