COMPREHENSIVE EDITION

PROGRAMMING AND PROBLEM SOLVING WITH C++

SIXTH EDITION

Nell Dale and Chip Weems

Chapter 3 + 4

Output and Input

Functions

- **Every C++ program must have a function called `main`**
- **Program execution always begins with function `main`**
- **Any other functions are subprograms and must be called by the `main` function**

```
void main(void) {
  int x;
  cout << "Pick a number from 1 to 10: ";
  cin >> x;
  printSquare(x);
  printParity(x);
}
void printSquare(int x) {
  cout << x
       << " squared is "
       << x * x << endl;
}
void printParity(x) {
  cout << x << " is " << x % 2
       << endl;
}
```

Function Calls

- One function calls another by using the name of the called function together with () containing an argument list
- temporarily transfers control from the calling function to the called function
- body block of function main should be concise
- Function calls are used to do subtasks
- Every C++ function has a return type

```
void main(void) {
  int x;
  cout << "Pick a number from 1 to 10: ";
  cin >> x;
  printSquare(x);
  printParity(x);
}
void printSquare(int x) {
  cout << x
       << " squared is "
       << x * x << endl;
}
void printParity(x) {
  cout << x << " is " << x % 2
       << endl;
}
```

Function call

Where are functions?

Functions are subprograms

- **located in libraries, or**
- **written by programmers for their use in a particular program**

HEADER FILE	FUNCTION	EXAMPLE OF CALL	VALUE
<cstdlib>	abs(i)	abs(-6)	6
<cmath>	pow(x,y)	pow(2.0,3.0)	8.0
	fabs(x)	fabs(-6.4)	6.4
<cmath>	sqrt(x)	sqrt(100.0)	10.0
	sqrt(x)	sqrt(2.0)	1.41421
<cmath>	log(x)	log(2.0)	.693147
<iomanip>	setprecision(n)	setprecision(3)	

Write C++ Expressions for

The square root of $b^2 - 4ac$

```
sqrt(b * b - 4.0 * a * c)
```

The square root of the average of myAge and yourAge

```
sqrt((myAge  +  yourAge) / 2)
```

Function Call Syntax

returnVal FunctionName (Argument List);

- **The argument list is a way for functions to communicate with each other by passing information**
- **The argument list can contain zero, one, or more arguments, separated by commas, depending on the function**
- **The function can return void (no value) or a value.**

```
void printPair(int x, int y);
int choose(int a, int b);
double getPosRoot(double a,
          double b, double c);
```

A void function call stands alone

```
#include <iostream>

void  DisplayMessage(int  n);
// Declares function

int main()
{
    DisplayMessage(15);
    // Function call
    cout  <<  "Good Bye"  <<  endl;
    return 0;
}
```

A void function does NOT return a value

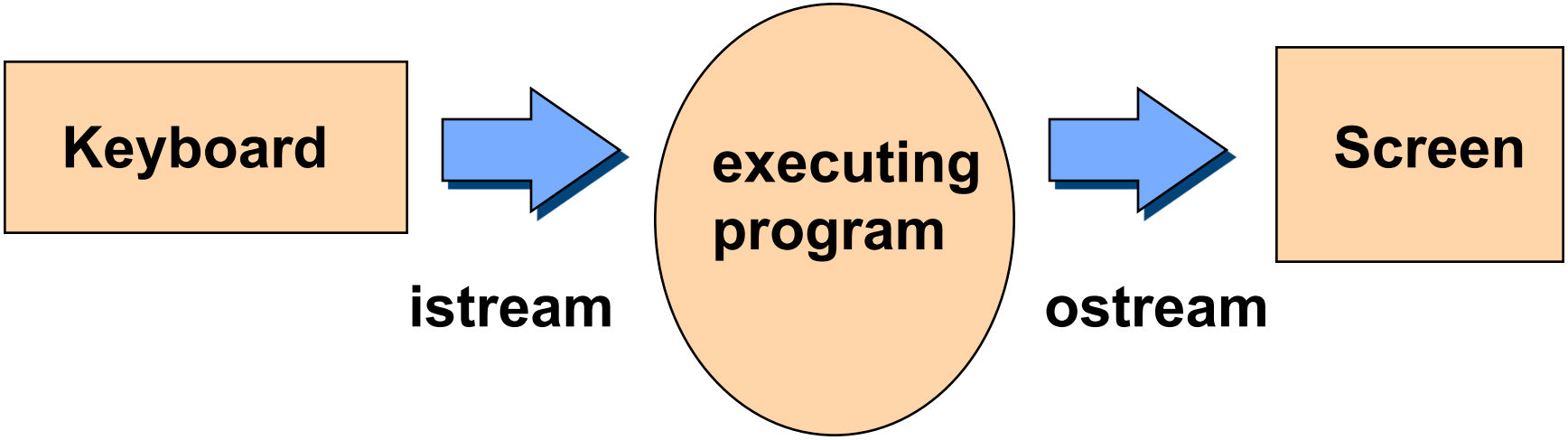
```
// Header and body here

void  DisplayMessage(int  n)
{
    cout  <<  "I have liked math for  "
          <<  n  <<  " years"  << endl;
}
```

Two Kinds of Functions

Value-Returning	Void
Always returns a single value to its caller and is called from within an expression	Never returns a value to its caller and is called as a separate statement

No I/O is built into C++

- Instead, a library provides input stream and output stream



<iostream> is header file

- For a library that defines 3 objects

  An istream object named cin (keyboard)

  An ostream object named cout (screen)

  An ostream object named cerr (screen)

Insertion Operator(<<)

- The insertion operator `<<` takes 2 operands
- The left operand is a stream expression, such as `cout`
- The right operand is an expression of simple type, a `string`, or a manipulator

>> Operator

>> is called the input or extraction operator

>> is a binary operator

>> is left associative

Expression **Has value**

```
cin  >> age
```

```
cin
```

Statement

```
cin  >>  age  >>  weight;
```

<< and >> are a binary operators

<< is called the output or insertion operator

<< is left associative

Expression	Has value
cout << age	cout

Statement

```
cout << "You are "  << age  <<  " years old\n";
```

Output and Input Statements

SYNTAX

cout << *Expression* << *Expression* . . .;

cin >> *Variable* >> *Variable* . . .;

Top and bottom yield the same result.

```
cout << "The answer is ";
cout << 3 * 4;
```

```
cin >> length;
cin >> width;
```

```
cout << "The answer is "
     << 3 * 4;
```

```
cin >> length >> width;
```

Manipulators

- Manipulators are used only in input and output statements
- `endl, fixed, showpoint, setw, and setprecision` are manipulators that can be used to control output format
- `endl` is use to terminate the current output line and create blank lines in output

Output Statements

SYNTAX(revised)

```
cout << ExpressionOrManipulator
     << ExpressionOrManipulator . . .;
```

Using Manipulators `Fixed` and `Showpoint`

- Use the following statement to specify that (for output sent to the cout stream) decimal format (not scientific notation) be used,
- and that a decimal point be included (even for floating values with 0 as fractional part)

```
cout << fixed << showpoint;
```

setprecision(n)

- Requires `#include <iomanip>` and appears in an expression using insertion operator(<<)
- If `fixed` has already been specified, argument `n` determines the number of places displayed after the decimal point for floating point values
- Remains in effect until explicitly changed by another call to `setprecision`

What is exact output?

```
#include  <iomanip> // For setw() and setprecision()
#include  <iostream>

using  namespace  std;

int main()
{
    float   myNumber  =  123.4587;
    cout  <<  fixed  <<   showpoint;
    // Use decimal format
    // Print decimal points
    cout  <<  "Number is "  <<  setprecision(3)
          <<  myNumber     <<  endl;

    return  0;
}
```

OUTPUT

Number is 123.459

Value is rounded if necessary to be displayed with exactly 3 places after the decimal point

Manipulator `setw`

- "Set width" lets us control how many **character positions** the next data item should occupy when it is output
- `setw` is only for formatting numbers and strings, not `char` type data

setw(n)

- Requires `#include <iomanip>` and appears in an expression using insertion operator(<<)
- Argument `n` is called the **fieldwidth specification**
- Argument `n` determines the number of character positions in which to display a right-justified number or string (not char data)

setw(n)

- The number of character positions used is expanded if n is too narrow
- "Set width" affects only the very next item displayed and is useful to align columns of output

A) What is exact output?

```
#include  <iomanip>                    // For setw()
#include  <iostream>
#include  <string>

using  namespace  std;
```

A) What is exact output?, cont...

```
int  main()
{
    int  myNumber    =  123;
    int  yourNumber  =  5;

    cout <<  setw(10)  <<  "Mine"
         <<  setw(10)  <<  "Yours"  << endl
         <<  setw(10)  <<  myNumber
         <<  setw(10)  <<  yourNumber << endl;

    return 0;
}
```

Output

position

```
12345678901234567890
```

```
      Mine      Yours
       123          5
```

Each is displayed right-justified and each is located in a total of 10 positions

B) What is exact output?

```
#include  <iomanip> // For setw() and setprecision()
#include  <iostream>

using  namespace  std;

int  main()
{
    float myNumber    =  123.4;
    float yourNumber  =  3.14159;
```

B) What is exact output, continued?

```
cout  <<  fixed  << showpoint;
    // Use decimal format; print decimal points
    cout  <<  "Numbers are: "  <<  setprecision(4)
          <<  endl  <<  setw(10) <<  myNumber
          <<  endl  <<  setw(10) <<  yourNumber
          <<  endl;
    return 0;
}
```

OUTPUT

```
1234567890123456789 0
Numbers are:
  123.4000
    3.1416
```

Each is displayed right-justified and rounded if necessary and each is located in a total of 10 positions with 4 places after the decimal point

More Examples

```
    float  x   =  312.0;
    float  y   =  4.827;

OUTPUT
    cout  << fixed << showpoint;

    cout  << setprecision(2)
          << setw(10) << x  << endl
           << setw(10) << y  << endl;

    cout  << setprecision(1)
          << setw(10) << x  << endl
           << setw(10) << y  << endl;

    cout  << setprecision(5)
          << setw(7) << x  << endl
           << setw(7) << y << endl;
```

312.0 4.827

x y

```
,,,, 312.00
 ,,,,,, 4.83

,,,,, 312.0
 ,,,,,,, 4.8

 312.00000
    4.82700
```

HEADER FILE	MANIPULATOR	ARGUMENT TYPE	EFFECT
<iostream>	endl	none	terminates output line
<iostream>	showpoint	none	displays decimal point
<iostream>	fixed	none	activates scientific notation
<iomanip>	setw(n)	int	sets fieldwidth to n positions
<iomanip>	setprecision(n)	int	sets precision to n digits

Input

Giving a Value to a Variable

In your program you can assign (give) a value to the variable by using the **assignment operator =**

```
ageOfDog = 12;
```

or by another method, such as

```
cout << "How old is your dog?";
cin  >> ageOfDog;
```

Extraction Operator (>>)

- Variable cin is predefined to denote an input stream from the standard input device (the keyboard)

- The extraction operator >> called "get from" takes 2 operands; the left operand is a stream expression, such as cin--the right operand is a variable of simple type

Extraction Operator (>>)

- Operator >> attempts to extract (inputs) the next item from the input stream and to store its value in the right operand variable
- >> "skips over" (actually reads but does not store anywhere) leading white space characters as it reads your data from the input stream(either keyboard or disk file)

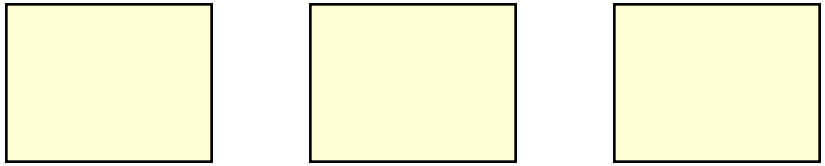
Whitespace Characters Include . . .

- blanks
- tabs
- end-of-line (newline) characters
- newline character created by:
  - hitting Enter or Return at the keyboard or
  - by using the manipulator `endl` or by using the symbols "\n" in the program

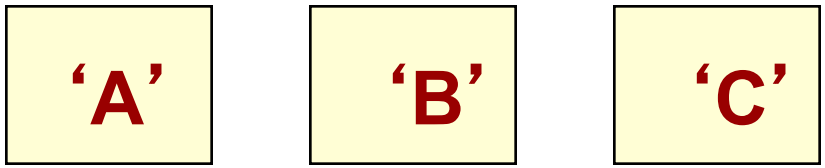
At keyboard you type:

A[space]B[space]C[Enter]

```
char   first;
char   middle;
char   last;
```




```
cin  >>   first ;
cin  >>   middle ;
cin  >>   last ;
```

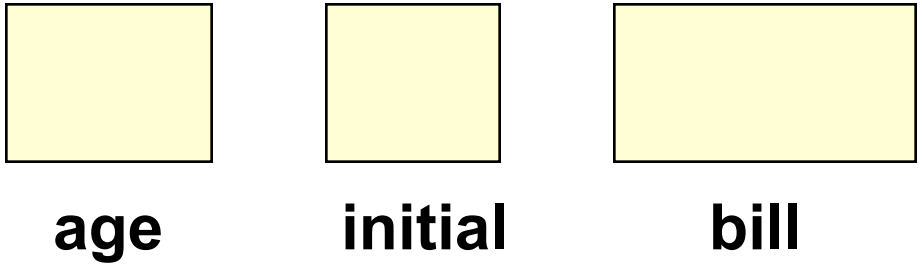


NOTE: A file reading marker is left pointing to the newline character after the 'C' in the input stream

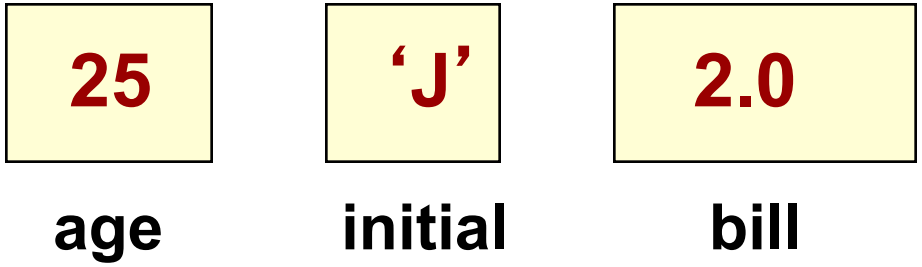
At keyboard you type:

[space]25[space]J[space]2[Enter]

```
int      age;
char   initial;
float   bill;
```



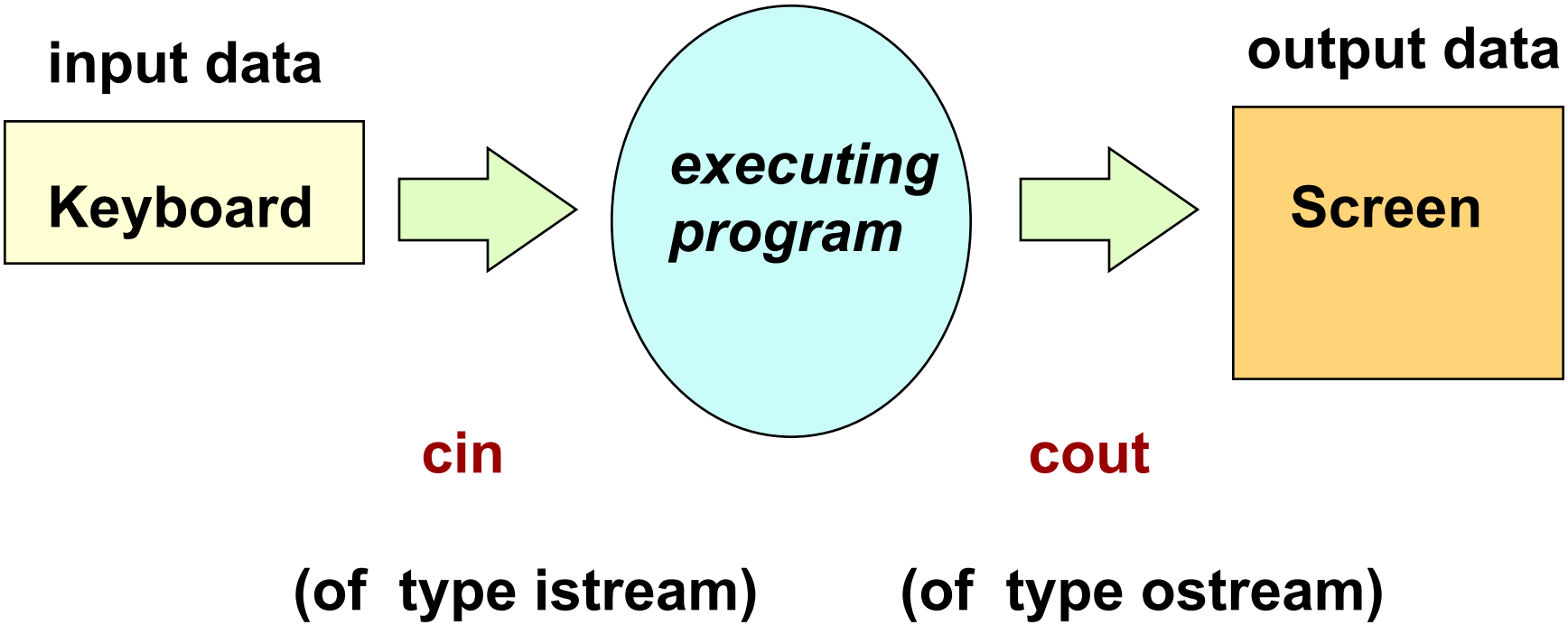

```
cin  >>  age;
cin  >>  initial;
cin  >> bill;
```



NOTE: A file reading marker is left pointing to the newline character after the 2 in the input stream

Keyboard and Screen I/O

```
#include <iostream>
```



Another example using >>

NOTE: shows the location of the file reading marker

STATEMENTS	CONTENTS	MARKER POSITION
int i; char ch; float x; cin >> i;	i: ; ch: ; x:	25 A\n 16.9\n
	i: 25 ; ch: ; x:	25 A\n 16.9\n
cin >> ch;	i: 25 ; ch: 'A' ; x:	25 A\n 16.9\n
cin >> x;	i: 25 ; ch: 'A' ; x: 16.9	25 A\n 16.9\n

Another Way to Read char Data

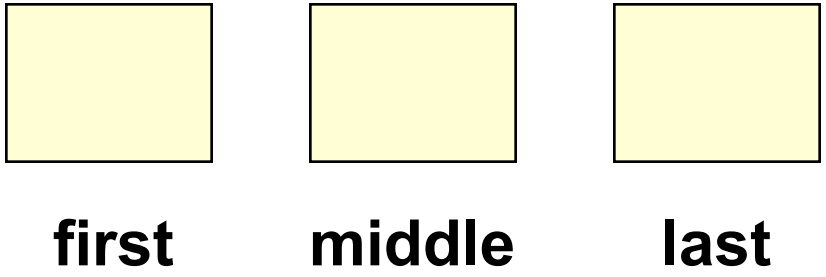
- The `get()` function can be used to read a single character.

- `get()` obtains the very next character from the input stream without skipping any leading whitespace characters

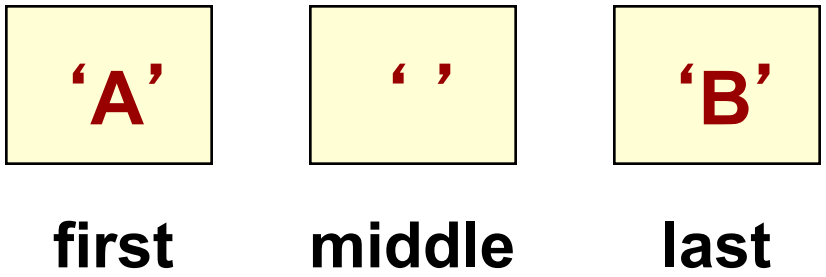
At keyboard you type:

`A[space]B[space]C[Enter]`

```
char   first;
char   middle;
char   last;
```




```
cin.get(first);
cin.get(middle);
cin.get(last);
```



NOTE: The file reading marker is left pointing to the space after the 'B' in the input stream

Use function ignore() to skip characters

The **ignore()** function is used to skip (read and discard) characters in the input stream

The call:

```
cin.ignore(howMany, whatChar);
```

will skip over up to **howMany** characters or until **whatChar** has been read, whichever comes first

An Example Using cin.ignore()

NOTE: ▯ shows the location of the file reading marker

STATEMENTS	CONTENTS	MARKER POSITION

`int a;`
`int b;`
`int c;`
	a: ▯ b: ▯ c: ▯	957 34 1235\n
		128 96\n

`cin >> a >> b;`	a: 957 b: 34 c: ▯	957 34 1235\n 128 96\n
`cin.ignore(100, '\n');`	a: 957 b: 34 c: ▯	957 34 1235\n 128 96\n
`cin >> c;`	a: 957 b: 34 c: 128	957 34 1235\n 128 96\n

Another Example Using cin.ignore()

NOTE: shows the location of the file reading marker

STATEMENTS	CONTENTS	MARKER POSITION
int i; char ch;	i: ; ch:	A 22 B 16 C 19\n
cin >> ch;	i: ; ch: 'A'	A 22 B 16 C 19\n
cin.ignore(100, 'B');	i: ; ch: 'A'	A 22 B 16 C 19\n
cin >> i;	i: 16 ; ch: 'A'	A 22 B 16 C 19\n

String Input in C++

Input of a string is possible using the extraction operator >>

Example

```
string   message;
cin  >>  message;
cout <<  message;
```

However . . .

>> Operator with Strings

Using the extraction operator(>>) to read input characters into a string variable

- The >> operator skips any leading whitespace characters such as blanks and newlines
- It then reads successive characters into the string
- >> operator then stops at the first trailing whitespace character (which is not consumed, but remains waiting in the input stream)

String Input Using >>

```
string   firstName;
string   lastName;
cin  >>  firstName >> lastName;
```

Suppose input stream looks like this:

□□Joe□Hernandez□23

What are the string values?

Results Using >>

```
string   firstName;
string   lastName;
cin  >>  firstName >> lastName;
```

Result

"Joe"	"Hernandez"
firstName	lastName

getline() Function

- Because the extraction operator stops reading at the first trailing whitespace, >> cannot be used to input a string with blanks in it
- Use the `getline` function with 2 arguments to overcome this obstacle
- First argument is an input stream variable, and second argument is a string variable

Example

```
string    message;
getline(cin,  message);
```

`getline(inFileStream, str)`

- `getline` does not skip leading whitespace characters such as blanks and newlines
- `getline` reads successive characters(including blanks) into the string, and stops when it reaches the newline character '\n'
- The newline is consumed by `getline`, but is not stored into the string variable

String Input Using getline

```
string    firstName;
string    lastName;
getline(cin,  firstName);
getline(cin,  lastName);
```

Suppose input stream looks like this:

□□Joe□Hernandez□23

What are the string values?

Results Using getline

```
string    firstName;
string    lastName;
getline(cin,  firstName);
getline(cin,  lastName);
```

" Joe Hernandez 23"	?
firstName	lastName

Interactive I/O

- In an interactive program the user enters information while the program is executing
- Before the user enters data, a prompt should be provided to explain what type of information should be entered
- The amount of information needed in the prompt depends on
  - the complexity of the data being entered, and
  - the sophistication of the person entering the data

Prompting for Interactive I/O

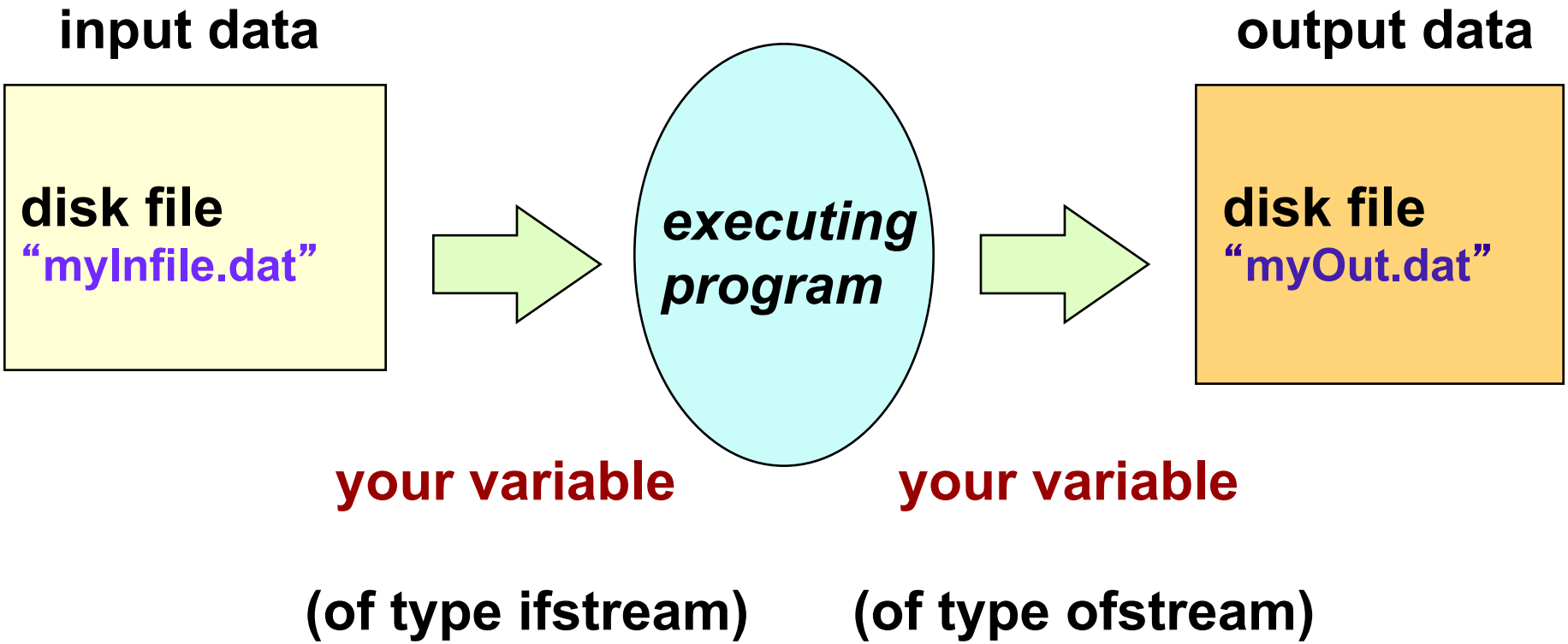
```
// Pattern: cout(prompt) cin(read value)
cout  <<  "Enter part number : "  <<  endl;
cin    >>  partNumber;
cout  <<  "Enter quantity ordered : "  <<
endl;
cin    >>  quantity;
cout  <<  "Enter unit price : " <<  endl;
cin    >>  unitPrice;
// Calculate and print results
```

Prompting for Interactive I/O, cont...

```
totalPrice  =  quantity * unitPrice;
cout  <<  "Part # " <<  partNumber  <<  endl;
cout  <<  "Quantity: "  <<  quantity
      <<  endl;
cout  <<  "Unit Cost: $ "  <<  setprecision(2)
      <<  unitPrice  <<  endl;
cout  <<  "Total Cost: $ " <<  totalPrice
      <<  endl;
```

Disk Files for I/O

```
#include <fstream>
```



Disk I/O

To use disk I/O

- Access #include <fstream>
- Choose valid identifiers for your file streams and declare them
- Open the files and associate them with disk names

Disk I/O, cont...

- Use your file stream identifiers in your I/O statements(using >> and << , manipulators, get, ignore)
- Close the files

Disk I/O Statements

```
#include <fstream>

ifstream   myInfile;          // Declarations
ofstream  myOutfile;

myInfile.open("myIn.dat"); // Open files
myOutfile.open("myOut.dat");

myInfile.close();             // Close files
myOutfile.close();
```

Opening a File

Opening a file

- Associates the C++ identifier for your file with the physical(disk) name for the file
  - If the input file does not exist on disk, open is not successful
  - If the output file does not exist on disk, a new file with that name is created
  - If the output file already exists, it is erased

Opening a File

Opening a file

- Places a file reading marker at the very beginning of the file, pointing to the first character in the file

Stream Fail State

- When a stream enters the **fail state**,
  - Further I/O operations using that stream have no effect at all
  - The computer does not automatically halt the program or give any error message

Stream Fail State

- **Possible reasons** for entering fail state include:
  - Invalid input data (often the wrong type)
  - Opening an input file that doesn't exist
  - Opening an output file on a disk that is already full or is write-protected

Run Time File Name Entry

```
#include <string>
// Contains conversion function c_str

ifstream  inFile;
string    fileName;

// Prompt:
cout << "Enter input file name: " << endl;
cin  >>  fileName;

// Convert string fileName to a C string type
inFile.open(fileName.c_str());
```

String functions

- length
- size
- find

`length` Function

- Function `length` returns an unsigned integer value that equals the number of characters currently in the string
- Function `size` returns the same value as function length
- You must use dot notation in the call to function `length` or `size`

`find` Function

- Function `find` returns an unsigned integer value that is the beginning position for the first occurrence of a particular substring within the string
- The substring argument can be a `string` constant, a `string` expression, or a `char` value
- If the substring was not found, function `find` returns the special value `string::npos`

`substr` Function

- Function `substr` returns a particular substring of a string
- The first argument is an unsigned integer that specifies a starting position within the string
- The second argument is an unsigned integer that specifies the length of the desired substring
- Positions of characters within a string are numbered starting from 0, not from 1

Project 1

- Start by making a skeleton
  - Create the java file
  - Create the main method
  - add comments for each part of the algorithm
  - fill in code as you learn
  - compile after each line or small set of changes.