C++ Programming Essentials

Unit 1: Sequential Programming

CHAPTER 4: PROGRAM INPUT AND THE SOFTWARE DESIGN PROCESS

DR. ERIC CHOU

IEEE SENIOR MEMBER

LECTURE 1

Basic Function Knowledge

Chapter 4 Topics

- Using Function Arguments
- Using C++ Library Functions in Expressions
- Calling a Void Function
- C++ Manipulators to Format Output
- String Operations length, find, substr

Chapter 4 Topics

- Input Statements to Read Values for a Program using >>, and functions **get, ignore, getline**
- Prompting for Interactive Input/Output
- Using Data Files for Input and Output
- Object-Oriented Design Principles
- Functional Decomposition Methodology

Function Concept in Math

Function definition

$$f(x) = 5x - 3$$

Name of function

Parameter of function

- When $x = 1$, $f(x) = 2$ is the returned value.
- When $x = 4$, $f(x) = 17$ is the returned value.
- Returned value is determined by the function definition and by the values of any parameters.

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

Functions

Function Calls

- one function calls another by using the name of the called function together with ( ) containing an argument list

- a function call temporarily transfers control from the calling function to the called function

What is in a block?

```
{
 // 0 or more statements here;

}
```

Every C++ function has 2 parts

```
int main ( )
{
  return 0;

}
```

heading

body block

Shortest C++ Program

type of returned value

name of function

```
int  main ( )
{

    return 0;

}
```

What is in a heading?

type of returned value

name of function

says no parameters

```
int main (      )
```

More About Functions

- it is not considered good practice for the body block of function main to be long
- function calls are used to do tasks
- every C++ function has a return type
- if the return type is not void, the function returns a value to the calling block

Where are functions?

located in libraries

OR

written by programmers

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 )
```

LECTURE 3

Transfer of Control

Function Call

- a function call temporarily transfers control to the called function's code

- when the function's code has finished executing, control is transferred back to the calling block

Function Call Syntax

FunctionName (Argument List)

- The argument list is a way for functions to communicate with each other by passing information.
- The argument list can contain 0, 1, or more arguments, separated by commas, depending on the function.

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.

LECTURE 4

I/O Stream and Operators (<<, >>)

<< is a binary operator

<< is called the output or insertion operator

<< is left associative

EXPRESSION	HAS VALUE
cout << age	cout

STATEMENT

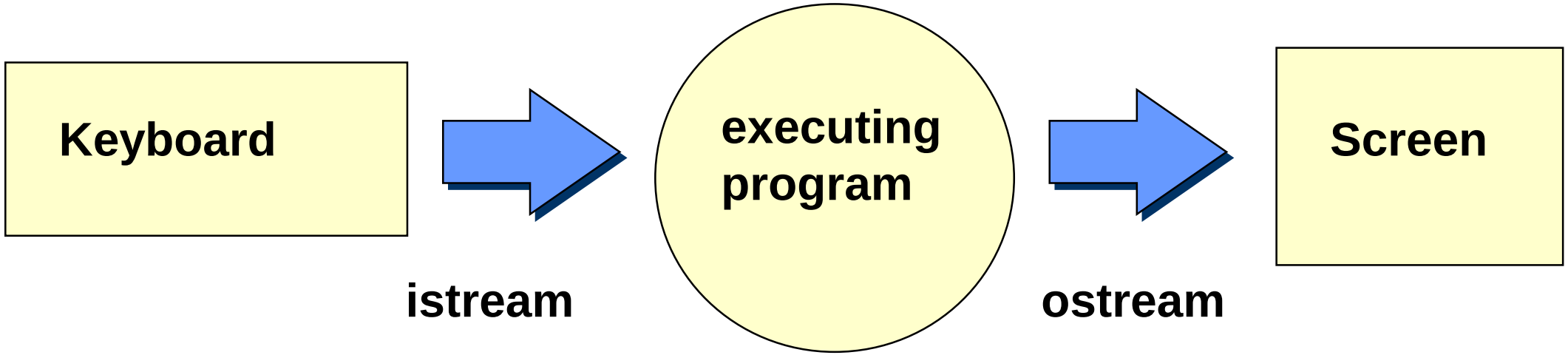
```
cout << "You are "  << age  <<  " years old\n" ;
```

<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)

No I/O is built into C++

- instead, a library provides input stream and output stream



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

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, or a string, or a manipulator

Output Statements

SYNTAX (revised)

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

Output Statements

SYNTAX

```
cout << Expression << Expression . . . ;
```

These examples yield the same output.

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

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

LECTURE 5

Output Formatting

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

Demo Program:

precision.cpp

Go Dev C++!!!

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, and determines the number of character positions in which to display a right-justified number or string (not char data). The number of 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

What is exact output?

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

using  namespace  std;

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

Demo Program:

precision2.cpp

Go Dev C++!!!

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch4\Precision2\Precision2.exe

```
     Mine      Yours
      123          5
```

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 ;

   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

```
12345678901234567890
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

Demo Program:

precision3.cpp

Go Dev C++!!!

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch4\Precision3\Precision3.exe

```
Numbers are:
  123.4000
    3.1416
```

More Examples

312.0

x

4.827

y

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

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 ;
```

OUTPUT

```
  ""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**	**suppresses scientific notation**
<iomanip>	**setw(n)**	**int**	**sets fieldwidth to n positions**
<iomanip>	**setprecision(n)**	**int**	**sets precision to n digits**

LECTURE 6

Other I/O Stream Functions

`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

1

function find returns an unsigned integer value that is the beginning position for the first occurrence of a particular substring within the string

2

the substring argument can be a string constant, a string expression, or a char value

3

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

What is exact output?

```
#include  <iostream>
#include  <string>        // for functions length, find, substr

using  namespace  std;

int  main (  )
{
    string  stateName = “Mississippi” ;

    cout  <<  stateName.length( )  << endl;

    cout  <<   stateName.find(“is”) <<  endl;

    cout <<  stateName.substr( 0, 4 ) <<  endl;

    cout << stateName.substr( 4, 2 ) <<  endl;

    cout << stateName.substr( 9, 5 ) <<  endl;

    return 0 ;
}
```

Demo Program:

precision4.cpp

Go Dev C++!!!

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch4\Precision4\Precision4.exe

```
11
1
Miss
is
pi
```

What is exact output?

```
#include  <iostream>
#include  <string>        // for functions length, find, substr

using  namespace  std;

int  main (  )
{
   string  stateName = “Mississippi” ;

   cout  <<  stateName.length( )  << endl;        // value 11

   cout  <<   stateName.find(“is”) <<  endl;      // value 1

   cout <<  stateName.substr( 0, 4 ) <<  endl;        // value “Miss”

   cout << stateName.substr( 4, 2 ) <<  endl;         // value “is”

   cout << stateName.substr( 9, 5 ) <<  endl;         // value “pi”

   return 0 ;
}
```

Demo Program:

precision5.cpp

Go Dev C++!!!

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch4\Precision5\Precision5.exe

```
11
1
Miss
is
pi
```

LECTURE 7

Demo Program: Map Measurement

Map Measurement Case Study

- You want a program to determine walking distances between 4 sights in the city.
- Your city map legend says one inch on the map equals 1/4 mile in the city.
- You use the measured distances between 4 sights on the map.
- Display the walking distances (rounded to the nearest tenth) between each of the 4 sights.

C++ Program

```
// ***********************************************
//  Walk program
//  This program computes the mileage (rounded to nearest
//  tenth of mile) for each of 4 distances, given map
//  measurements on map with scale of 1 in = 0.25 mile
// ***********************************************

#include <iostream>     // for cout, endl
#include <iomanip>      // For setprecision

using namespace std;

float  RoundToNearestTenth( float );  // declare function

const  float  SCALE = 0.25;     // Map scale (mi. per inch)
```

C++ Code Continued

```
const  float  DISTANCE1 = 1.5;  // First map distance
const  float  DISTANCE2 = 2.3;  // Second map distance
const  float  DISTANCE3 = 5.9;  // Third map distance
const  float  DISTANCE4 = 4.0;  // Fourth map distance


int  main( )
{
     float     totMiles;          // Total of rounded miles
     float     miles;      // One rounded mileage

     cout << fixed << showpoint  // Set output format
      << setprecision(1);

     totMiles = 0.0;       // Initialize total miles
```

```
// Compute miles for each distance on map

miles = RoundToNearestTenth( DISTANCE1 * SCALE );

cout <<  DISTANCE1 << " inches on map is "
 <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;


miles = RoundToNearestTenth( DISTANCE2 * SCALE );

cout <<  DISTANCE2 << " inches on map is "
 <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;
```

```
// Compute miles for other distances on map

miles = RoundToNearestTenth( DISTANCE3 * SCALE );

cout <<  DISTANCE3 << " inches on map is "
 <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;


miles = RoundToNearestTenth( DISTANCE4 * SCALE );

cout <<  DISTANCE4 << " inches on map is "
 <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;
```

```
     cout <<  endl << "Total walking mileage is  "
          <<  totMiles  <<  " miles."  << endl;

     return 0 ;          //  Successful completion
}

// ***********************************************

float  RoundToNearestTenth ( /* in */  float floatValue)

//  Function returns floatValue rounded to nearest tenth.

{
     return  float(int(floatValue * 10.0 + 0.5)) / 10.0;
}
```

Demo Program:

measurement.cpp

Go Dev C++!!!

LECTURE 8

Input Section

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;
```

>> is a binary operator

>> is called the input or extraction operator

>> is left associative

EXPRESSION	HAS VALUE
cin >> age	cin

STATEMENT

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

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.
- operator >> attempts to extract the next item from the input stream and store its value in the right operand variable

Input Statements

SYNTAX

```
cin >> Variable >> Variable . . . ;
```

These examples yield the same result.

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

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

Extraction Operator >>

- "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)

Input Examples

Syntax:

```
cin >> var1 >> var2 >> var3 ...  ;
```

Example: assume the input stream contains the following sequence of characters:

```
12  17  -19  4
```

int X, Y, Z; cin >> X >> Y >> Z;	int X, Y, Z, W; cin >> X >> Y >> Z; cin >> W;

In both examples, assuming they're done independently, the result is

X = 12, Y = 17, and Z = -19. Also, in the second, W = 4.

Extraction operator >>

- When 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, and stops at the first trailing whitespace character (which is not consumed, but remains waiting in the input stream)

Extraction Operator & Whitespace

Whitespace characters:

Name	Code
Newline	\n
Tab	\t
Blank	(space)
Carriage return	\r
Vertical tab	\v

Extraction operator will ignore whitespace

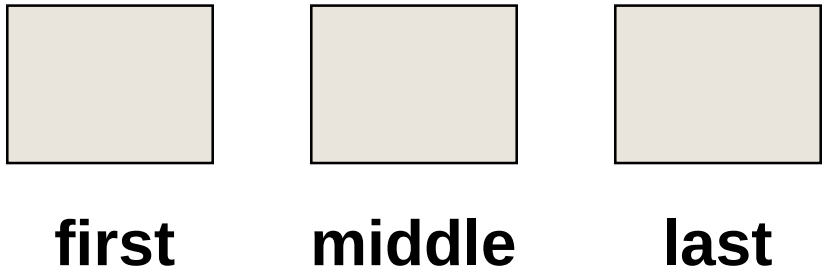
LECTURE 9

Input Example

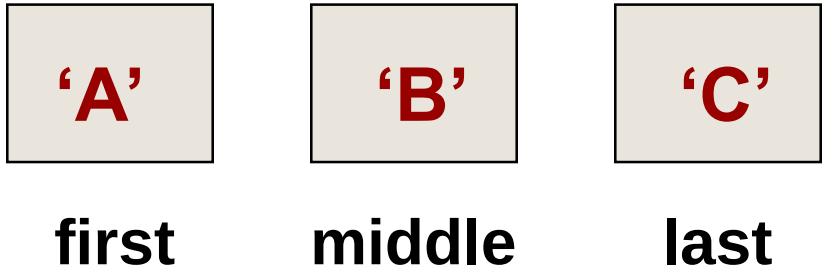
File contains:

`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.

File contains:

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

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

age	initial	bill

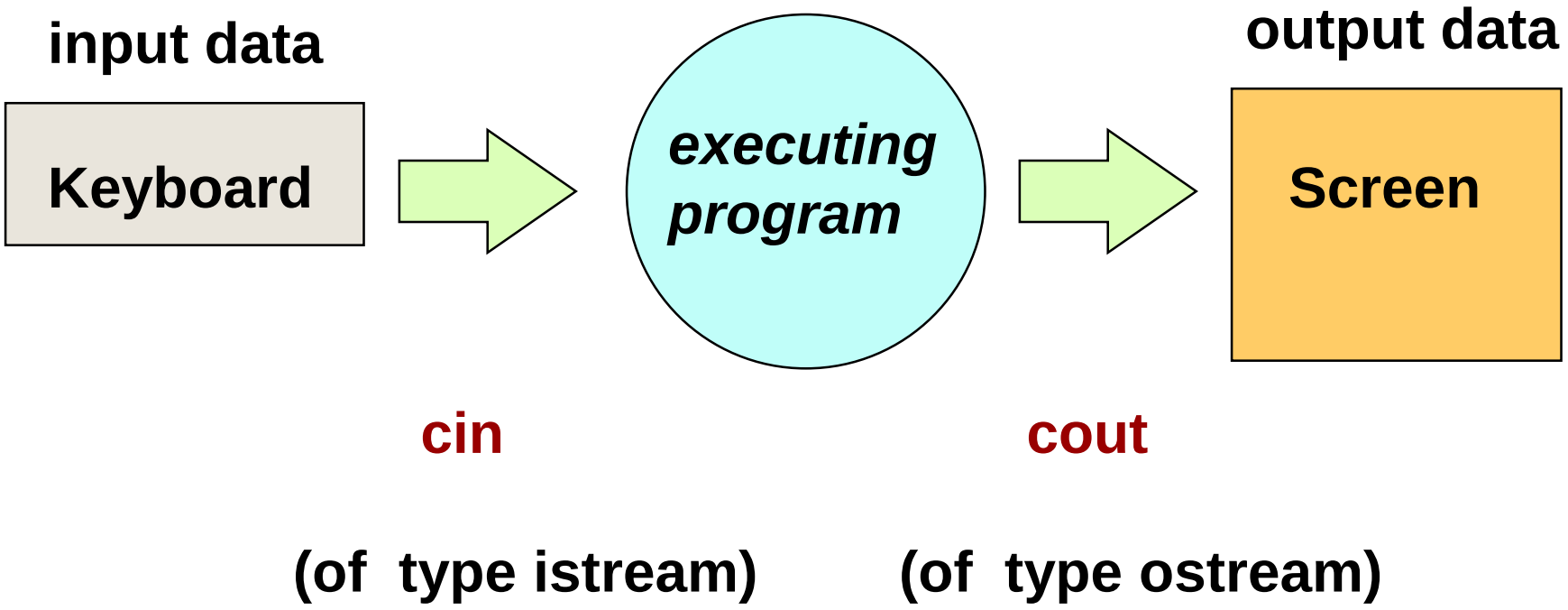
```
cin  >>  age ;
cin  >>  initial ;
cin  >> bill ;
```

age	initial	bill
25	‘J’	2.0

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

LECTURE 10

get(), ignore() Function and >> operator

Another Way to Read char Data

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

- It obtains the very next character from the input stream without skipping any leading whitespace characters.

File contains:

```
A B\n
```

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

first	middle	last

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

first	middle	last
‘A’	‘ ’	‘B’

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

get( ) Member Function

- To call a member function of an object, state the name of the object, followed by a period, followed by the function call:

```
cin.get(someChar);    // where someChar is a char variable
```

- This call to the `get( )` function will remove the next character from the stream `cin` and place it in the variable `someChar`.

A M

- So to read all three characters form we could have:

```
cin >> ch1;          // read 'A'
cin.get(someChar);   // read the space
cin >> ch2;          // read 'M'
```

- We could also have used the `get( )` function to read all three characters.

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.

Use function ignore( ) to skip characters

```
cin.ignore(80, '\n');
```

says to skip the next 80 input characters or to skip characters until a newline character is read, whichever comes first.

the ignore function can be used to skip a specific number of characters or halt whenever a given character occurs:

```
cin.ignore(100, '\t');
```

means to skip the next 100 input characters, or until a tab character is read, or whichever comes first.

An Example Using cin.ignore( )

NOTE: shows the location of the file reading marker

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

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

"J o e"	"Hernandez"
firstName	lastName

LECTURE 11

getline() Function

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 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 get, 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 );
```

firstName	lastName

LECTURE 12

I/O Streams

Interactive I/O

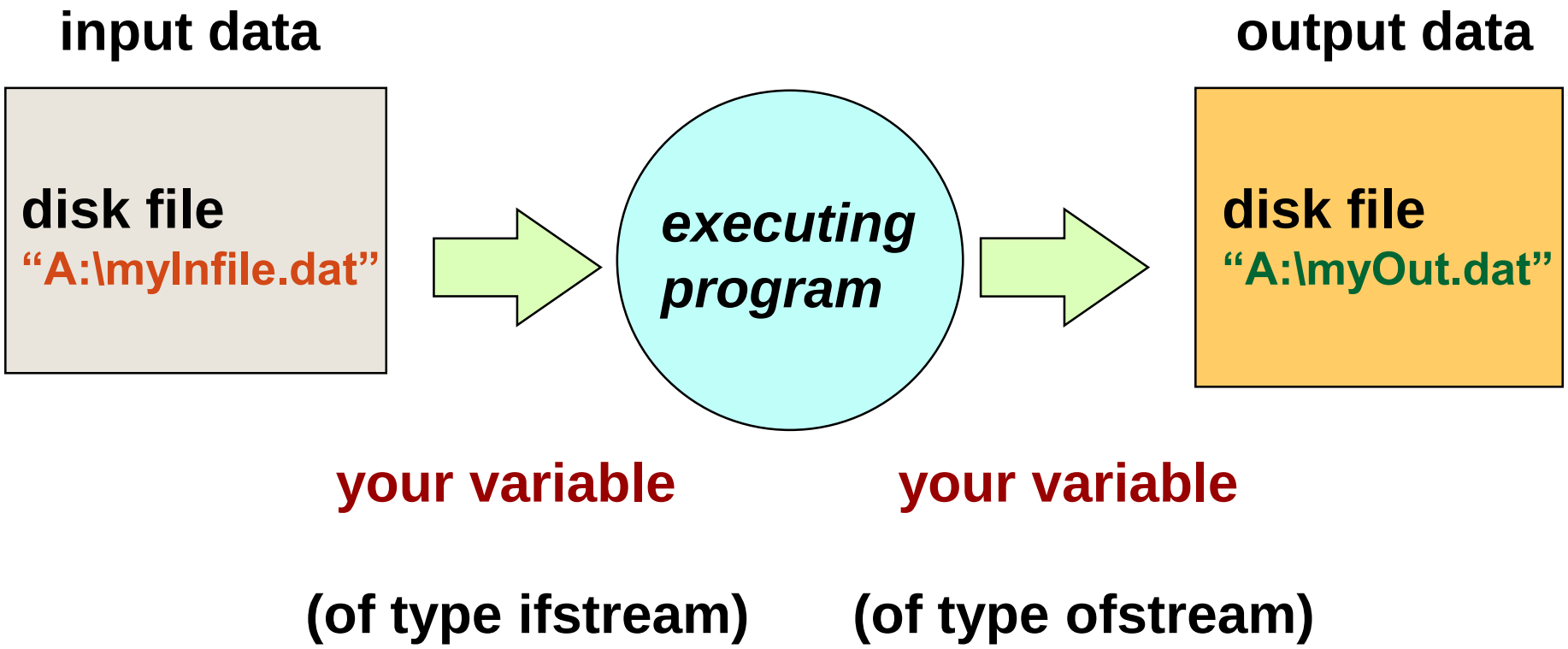
- in an interactive program the user enters information while the program is executing
- before the user enters data, a prompt should be provided to explain what type of information should be entered
- after the user enters data, the value of the data should be printed out for verification. This is called echo printing
- that way, the user will have the opportunity to check for erroneous data

Prompting for Interactive I/O

```
cout << "Enter part number : " << endl ;          // prompt
cin  >> partNumber ;

cout << "Enter quantity ordered : " << endl ;
cin  >> quantity ;

cout << "Enter unit price : " << endl ;
cin  >> unitPrice ;
totalPrice = quantity * unitPrice ;                // calculate

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

Diskette Files for I/O

#include <fstream>



use

#include <fstream>

- choose valid identifiers for your filestreams and declare them

- open the files and associate them with disk names
- use your filestream identifiers in your I/O statements (using >> and << , manipulators, get, ignore)
- close the files

To Use Disk I/O, you must

Statements for Using Disk I/O

```
#include <fstream>

ifstream   myInfile;                    // declarations
ofstream  myOutfile;

myInfile.open("A:\\myIn.dat");          // open files
myOutfile.open("A:\\myOut.dat");

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

What does opening a file do?

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

LECTURE 13

Case Study: Map Measurement

Map Measurement Case Study

- You want a program to determine walking distances between 4 sights in the city.
- Your city map legend says one inch on the map equals 1/4 mile in the city.
- Read from a file the 4 measured distances between sights on the map and the map scale.
- Output to a file the rounded (to the nearest tenth) walking distances between the 4 sights.

Using File I/O

```
// ***********************************************
//  Walk program using file I/O
//  This program computes the mileage (rounded to nearest
//  tenth of mile) for each of 4 distances, using input
//  map measurements and map scale.
// ***********************************************

#include <iostream>     // for cout, endl
#include <iomanip>      // for setprecision
#include <fstream>      // for file I/O

using namespace std;

float  RoundToNearestTenth( float );  // declare function
```

```
int  main( )
{
     float     distance1;      // First map distance
     float     distance2;      // Second map distance
     float     distance3;      // Third map distance
     float     distance4;      // Fourth map distance
     float     scale;             // Map scale (miles/inch)

     float     totMiles;          // Total of rounded miles
     float     miles;      // One rounded mileage

     ifstream  inFile;         // First map distance
     ofstream  outFile;        // Second map distance

     outFile << fixed << showpoint  // output file format
         << setprecision(1);

                // Open the files
     inFile.open("walk.dat");

     outFile.open("results.dat");
```

```
            // Get data from file

inFile >>  distance1  >> distance2  >> distance3
       >>  distance4  >> scale;

totMiles = 0.0;       // Initialize total miles

  // Compute miles for each distance on map

miles = RoundToNearestTenth( distance1 * scale );

outFile <<  distance1 << " inches on map is "
    <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;
```

```
miles = RoundToNearestTenth( distance2 * scale );

outFile <<  distance2 << " inches on map is "
    <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;

miles = RoundToNearestTenth( distance3 * scale );

outFile <<  distance3 << " inches on map is "
    <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;


miles = RoundToNearestTenth( distance4 * scale );

outFile <<  distance4 << " inches on map is "
    <<  miles  << " miles in city." << endl;

totMiles = totMiles + miles;
```

```
    //  Write total miles to output file

    outFile <<  endl << "Total walking mileage is  "
            <<  totMiles  <<  " miles."  << endl;

    return 0 ;          //  Successful completion
}

// ***********************************************

float  RoundToNearestTenth ( /* in */  float floatValue)

//  Function returns floatValue rounded to nearest tenth.

{
    return  float(int(floatValue * 10.0 + 0.5)) / 10.0;
}
```

Demo Program:

mapmeasurement.cpp

Go Dev C++!!!

```
3.2 inches on map is 3.2 miles in city.
2.8 inches on map is 2.8 miles in city.
0.6 inches on map is 0.6 miles in city.
4.0 inches on map is 4.0 miles in city.

Total walking mileage is  10.6 miles.
```

Stream Fail State

- when a stream enters the fail state, further I/O operations using that stream have no effect at all. But the computer does not automatically halt the program or give any error message
- 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 diskette that is already full or is write-protected

Entering File Name at Run Time

```
#include  <string>                    // contains conversion function c_str
ifstream    inFile;
string       fileName;
cout  <<  "Enter input file name : "  <<  endl ;          // prompt
cin    >>  fileName ;
                                // convert string fileName to a C string type
inFile.open( fileName.c_str( ) );
```

Functional Decomposition

- A technique for developing a program in which the problem is divided into more easily handled subproblems, the solutions of which create a solution to the overall problem.
- In functional decomposition, we work from the abstract (a list of the major steps in our solution) to the particular (algorithmic steps that can be translated directly into code in C++ or another language).

Functional Decomposition

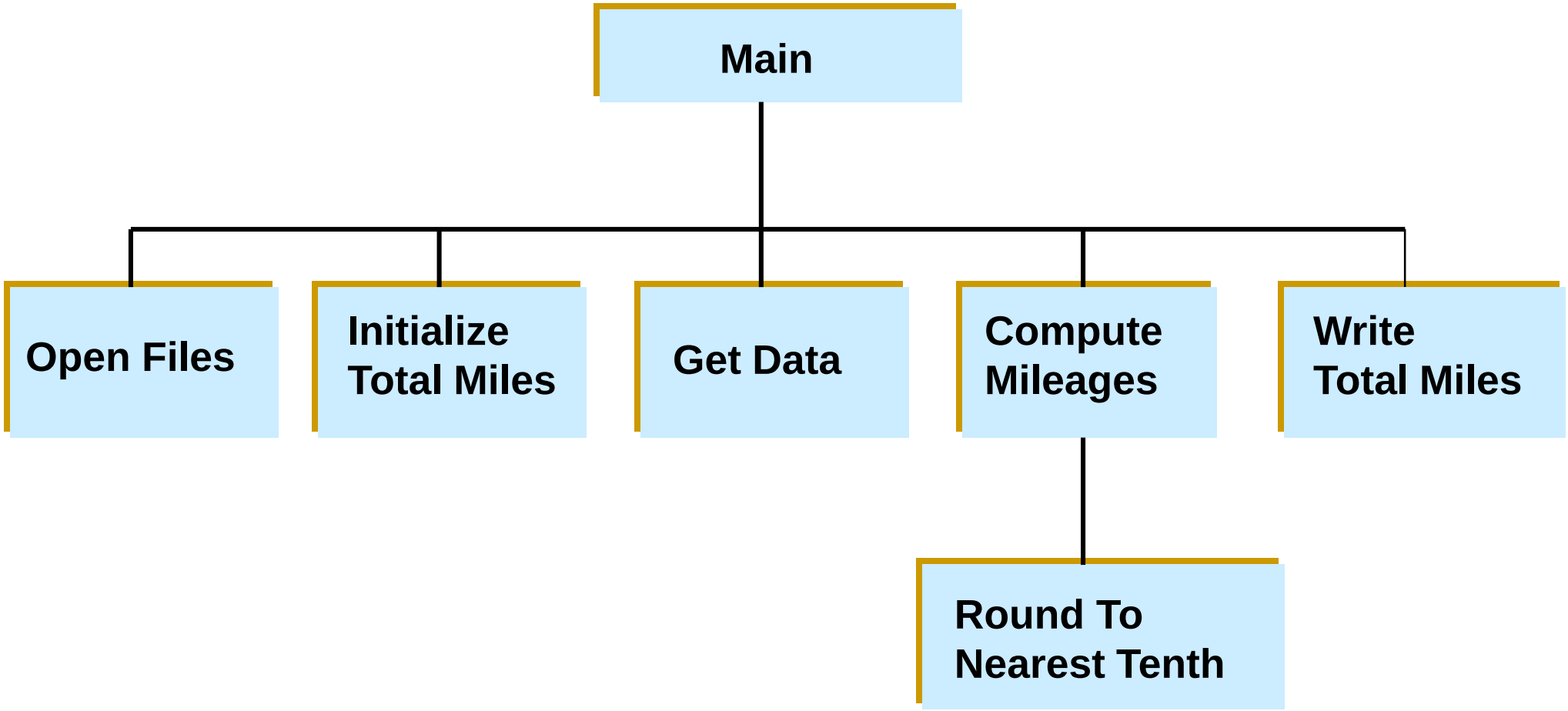
FOCUS is on actions and algorithms.

BEGINS by breaking the solution into a series of major steps. This process continues until each subproblem cannot be divided further or has an obvious solution.

UNITS are *modules* representing algorithms. A module is a collection of concrete and abstract steps that solves a subproblem. A module structure chart (hierarchical solution tree) is often created.

DATA plays a secondary role in support of actions to be performed.

Module Structure Chart

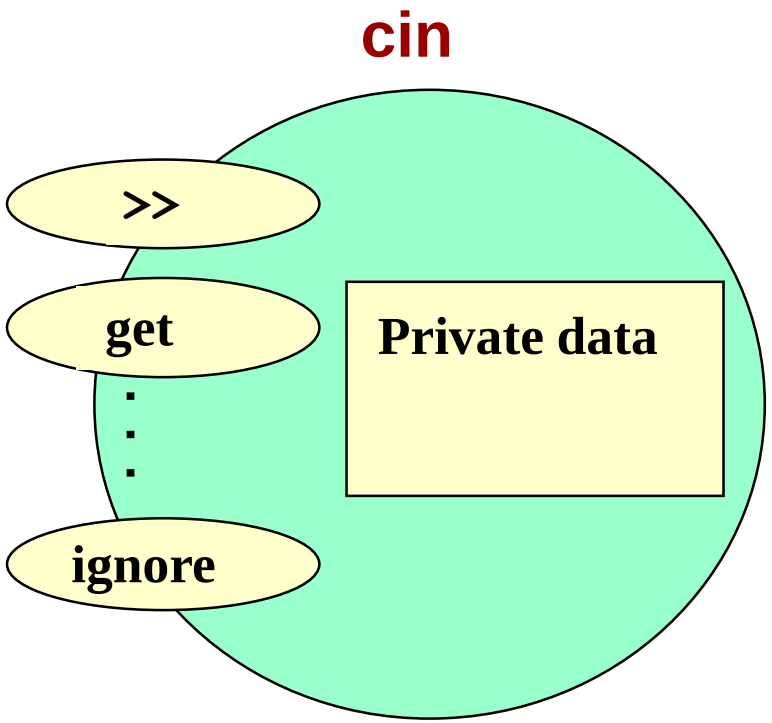


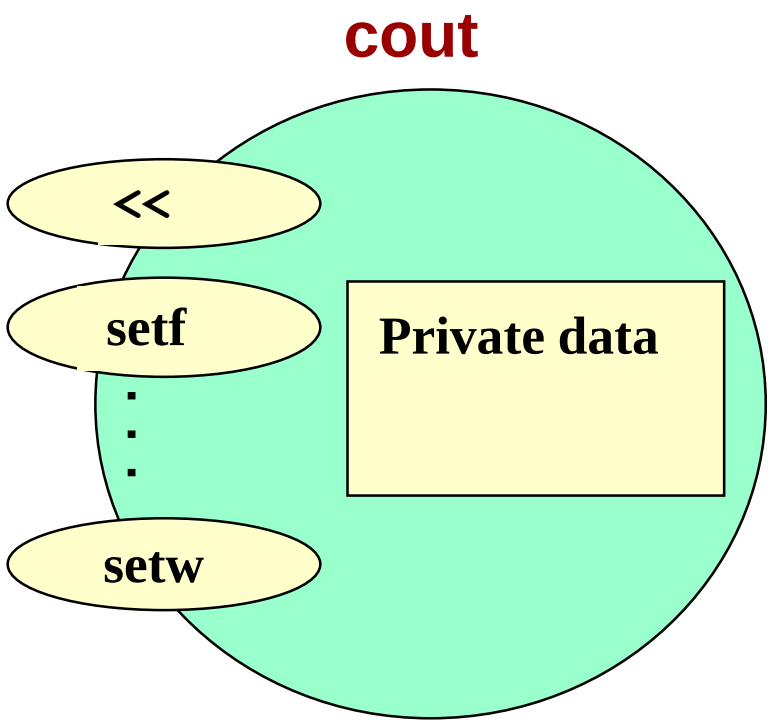
LECTURE 14

Object-Oriented Design

Object-Oriented Design

- A technique for developing a program in which the solution is expressed in terms of objects -- self- contained entities composed of data and operations on that data.





More about OOD

- languages supporting OOD include: C++, Java, Smalltalk, Eiffel, CLOS, and Object-Pascal
- a *class* is a programmer-defined data type and objects are variables of that type
- in C++, cin is an object of a data type (class) named istream, and cout is an object of a class ostream. Header files iostream and fstream contain definitions of stream classes
- a class generally contains private data and public operations (called *member functions*)

Object-Oriented Design (OOD)

FOCUS is on entities called objects and operations on those objects, all bundled together.

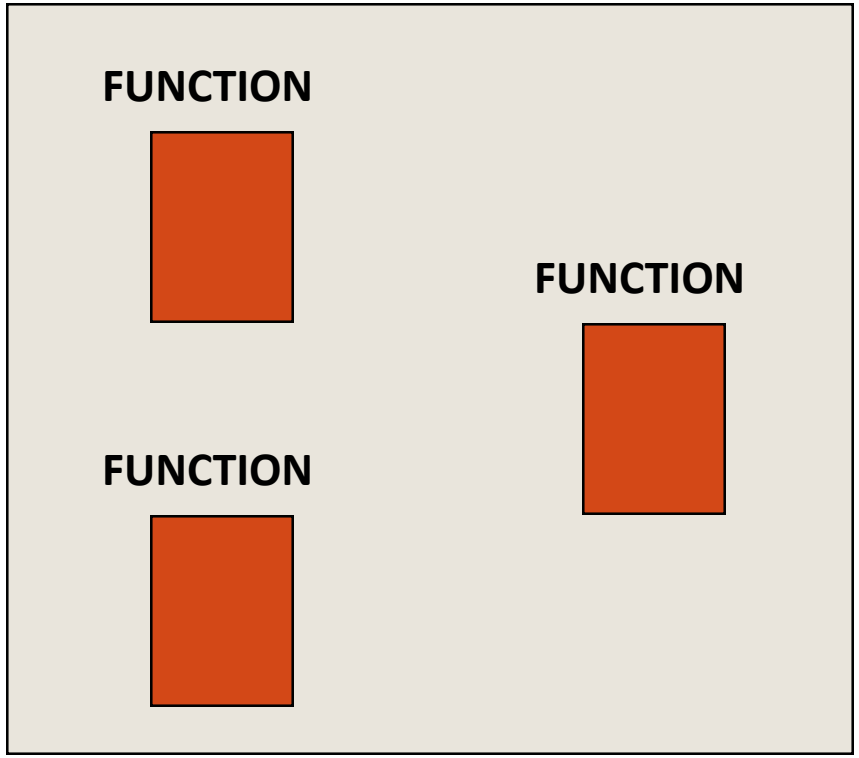
BEGINS by identifying the major objects in the problem, and choosing appropriate operations on those objects.

UNITS are *objects*. Programs are collections of objects that communicate with each other.

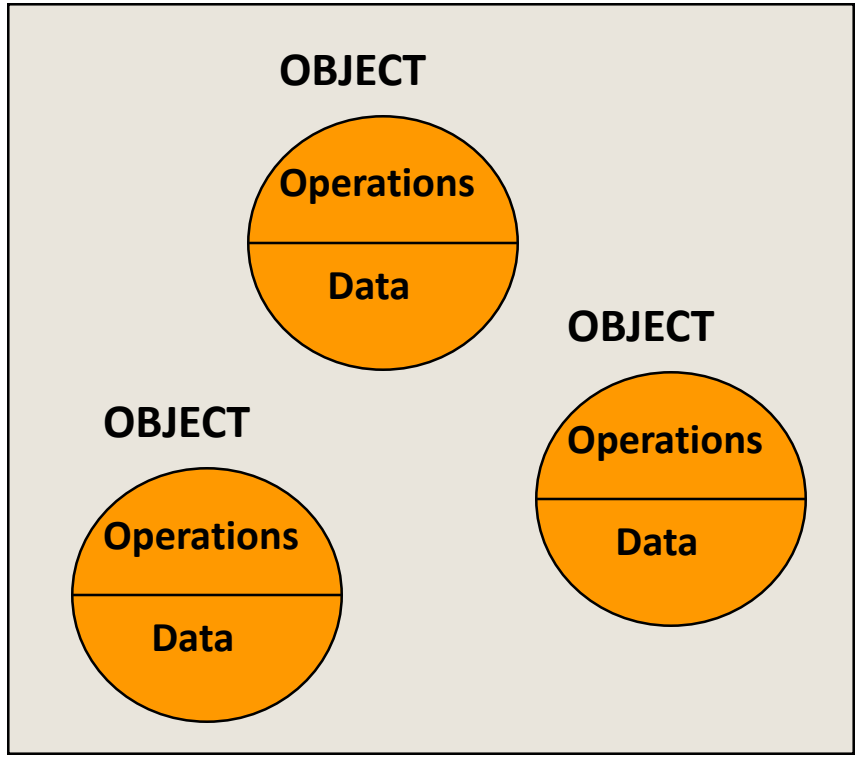
DATA plays a leading role. Algorithms are used to implement operations on the objects and to enable interaction of objects with each other.

Two Programming Methodologies

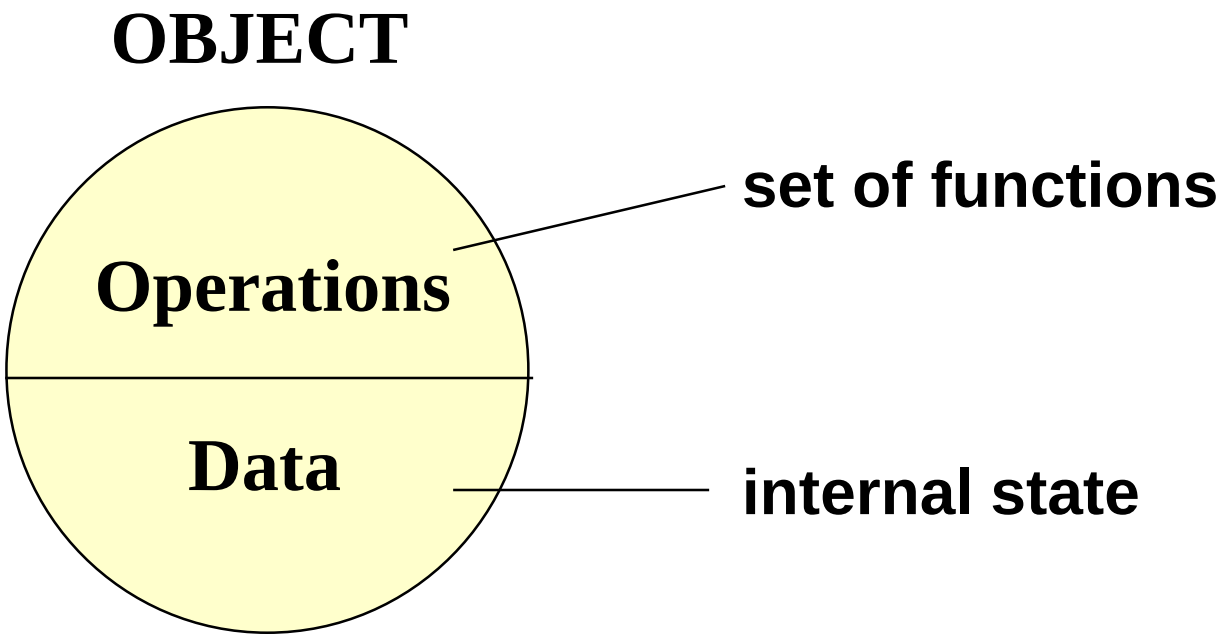
Functional Decomposition



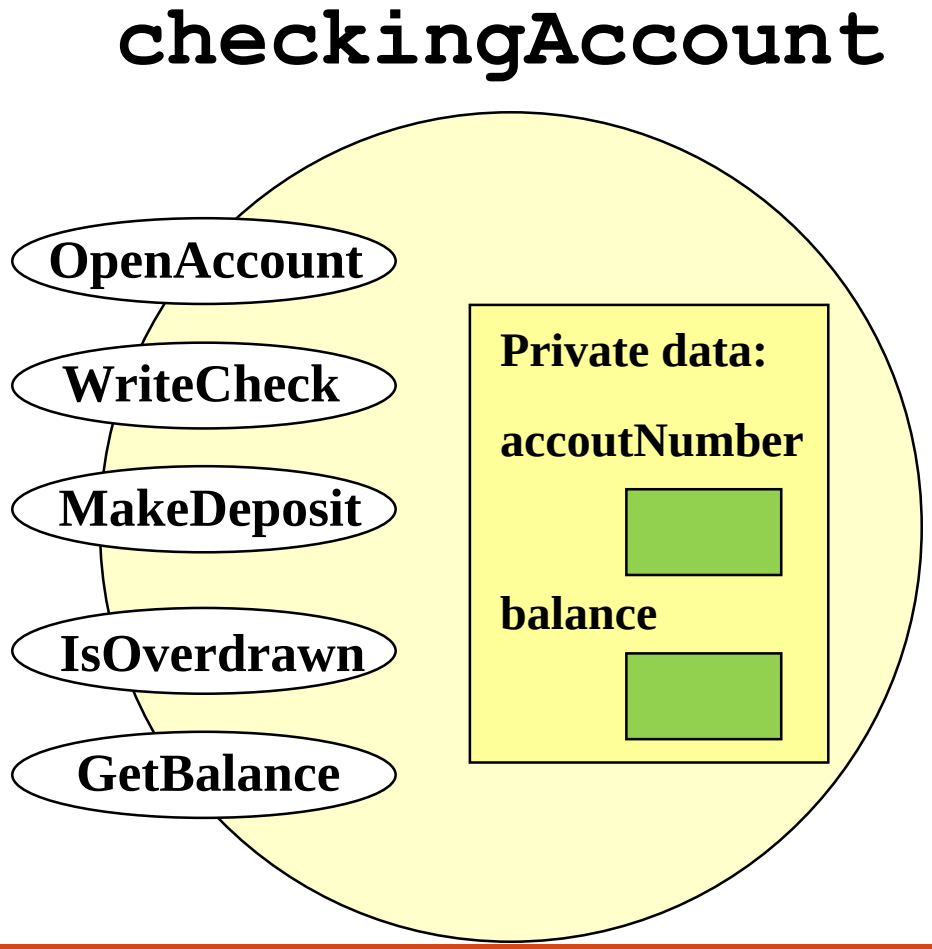
Object-Oriented Design



What is an object?



An object contains data and operations



Why use OOD with large software projects?

- objects within a program often model real-life objects in the problem to be solved
- many libraries of pre-written classes and objects are available as-is for re-use in various programs
- the OOD concept of inheritance allows the customization of an existing class to meet particular needs without having to inspect and modify the source code for that class--this can reduce the time and effort needed to design, implement, and maintain large systems

LECTURE 15

Case Study: Company Payroll

Company Payroll Case Study

- A small company needs an interactive program to figure its weekly payroll.
- The payroll clerk will input data for each employee.
- Each employee's wages and data should be saved in a secondary file.
- Display the total wages for the week on the screen.

Algorithm for Company Payroll Program

Initialize total company payroll to 0.0

Repeat this process for each employee

1. Get the employee's ID empNum
2. Get the employee's hourly payRate
3. Get the hours worked this week
4. Calculate this week's wages
5. Add wages to total company payroll
6. Write empNum, payRate, hours, wages to file

Write total company payroll on screen.

Company Payroll Program

```
// ***********************************************
//  Payroll program
//  This program computes each employee's wages and
//  the total company payroll
// ***********************************************

#include <iostream>     // for keyboard/screen I/O
#include <fstream>      // for file I/O

using namespace std;

void  CalcPay ( float,  float,  float& ) ;

const  float  MAX_HOURS = 40.0;  // Maximum normal hours
const  float  OVER_TIME = 1.5;    // Overtime pay factor
```

C++ Code Continued

```
int  main( )
{
     float      payRate;            //  Employee's pay rate
     float      hours;       //  Hours  worked
     float      wages;       //  Wages earned
     float      total;       //  Total company payroll
     int        empNum;      //  Employee  ID  number
     ofstream   payFile;     // Company  payroll file

     payFile.open( "payfile.dat" );  // Open file
     total = 0.0;                // Initialize total
```

```
cout << "Enter employee number: ";   // Prompt
cin  >> empNum;                      // Read ID number

while ( empNum  !=  0 )              // While not done
{
 cout << "Enter pay rate: ";
 cin  >> payRate ;                 // Read pay rate
 cout << "Enter hours worked: ";
 cin  >> hours ;                   // and hours worked

 CalcPay(payRate, hours, wages); // Compute wages

 total = total + wages;        // Add to total

 payFile << empNum << payRate
            << hours << wages << endl;

 cout << "Enter employee number: ";
 cin  >> empNum;                   // Read ID number
}
```

```
     cout <<  "Total payroll is  "
          <<  total  << endl;

     return 0 ;          //  Successful completion
}

// ***********************************************

void  CalcPay ( /* in */   float   payRate ,
                /* in */   float   hours ,
                /* out */  float&  wages )

//  CalcPay computes wages from the employee's pay rate
//  and the hours worked, taking overtime into account

{
     if ( hours  >  MAX_HOURS )
      wages = (MAX_HOURS * payRate ) +
                  (hours - MAX_HOURS) * payRate * OVER_TIME;
     else
      wages  =  hours * payRate;
}
```

Demo Program:

payroll.cpp

Go Dev C++!!!

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch4\Payroll\Payroll.exe

```
Enter employee number: 1
Enter pay rate: 20
Enter hours worked: 20
Enter employee number: 2
Enter pay rate: 30
Enter hours worked: 30
Enter employee number: 3
Enter pay rate: 20
Enter hours worked: 20
Enter employee number: 4
Enter pay rate: 40
Enter hours worked: 10
Enter employee number: 0
Total payroll is  2100
```