Chapter 13

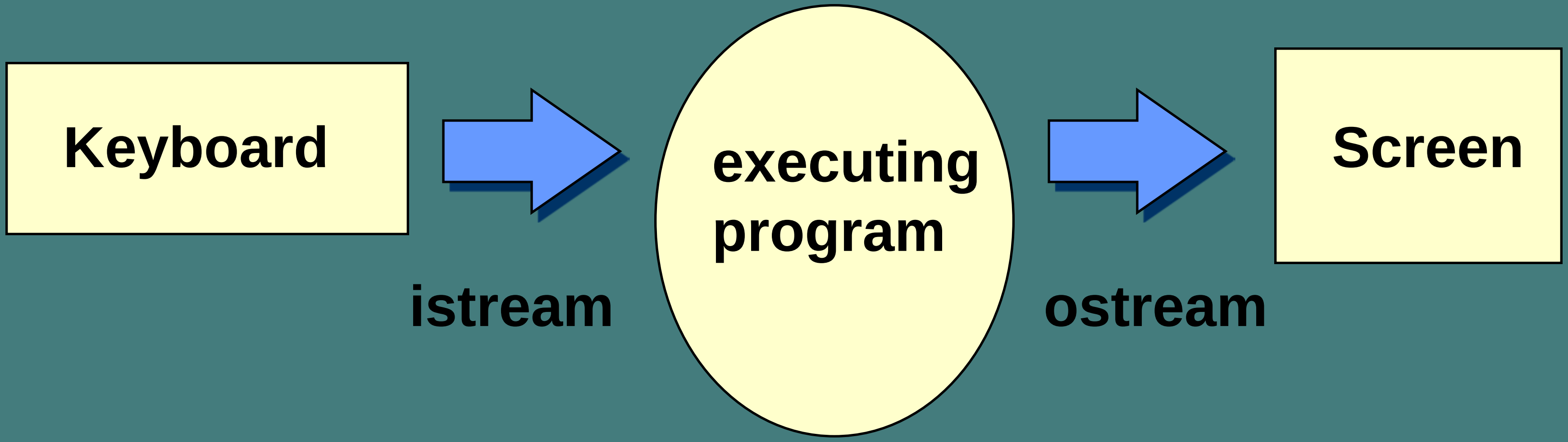
Input/Output Stream

Chapter 13 Topics(part 1)

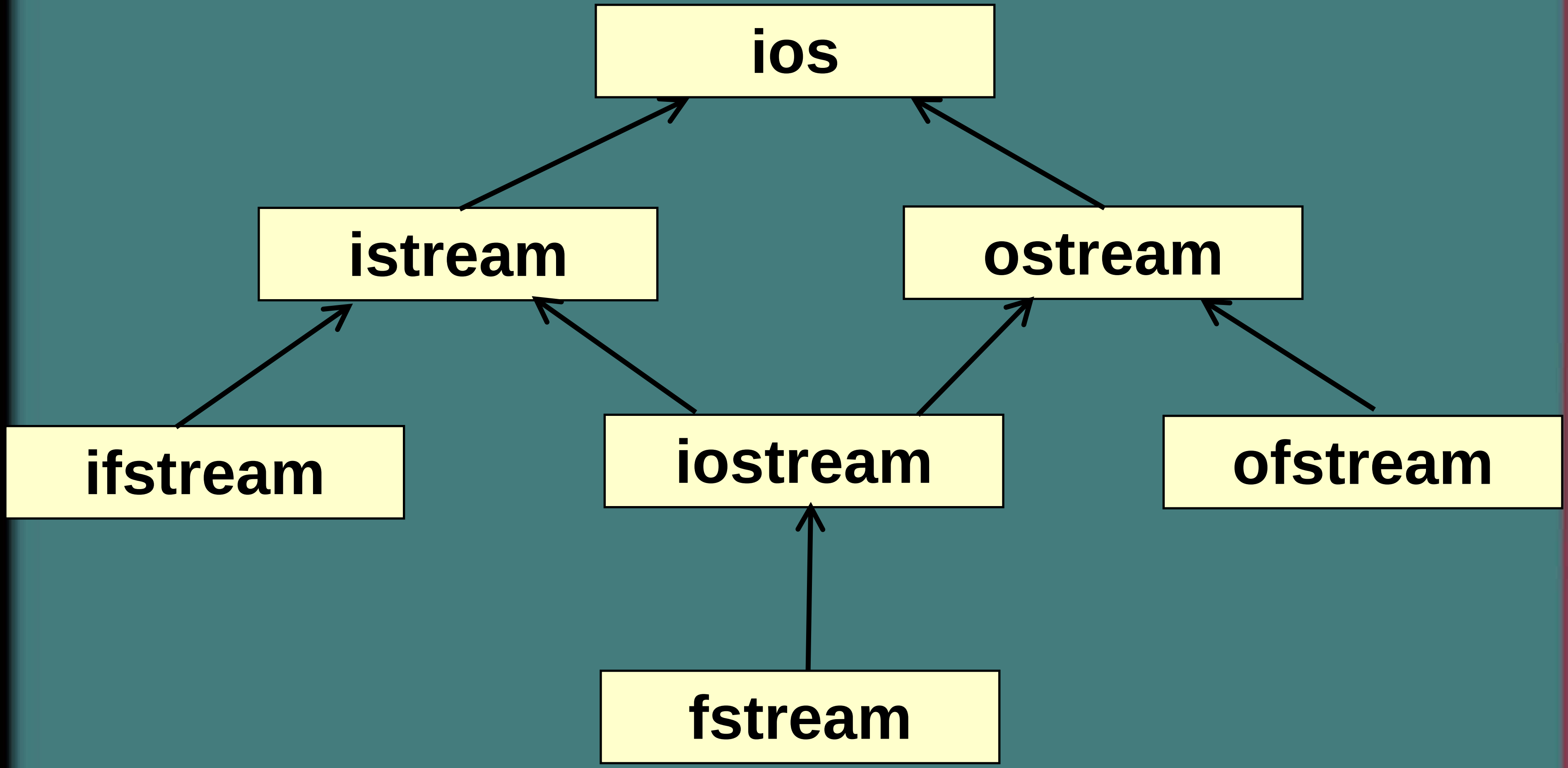
- Input/Output Stream in C++
  - Classes in I/O Library
  - Prompting for Interactive I/O
- Input Statements to Read Values for a Program
  - Using “>>”
  - Using Functions `get( )`
  - Using Functions `ignore( )`
  - Using Functions `getline( )`

No I/O is built into C++

- instead, a library provides input stream and output stream



Classes in I/O Library



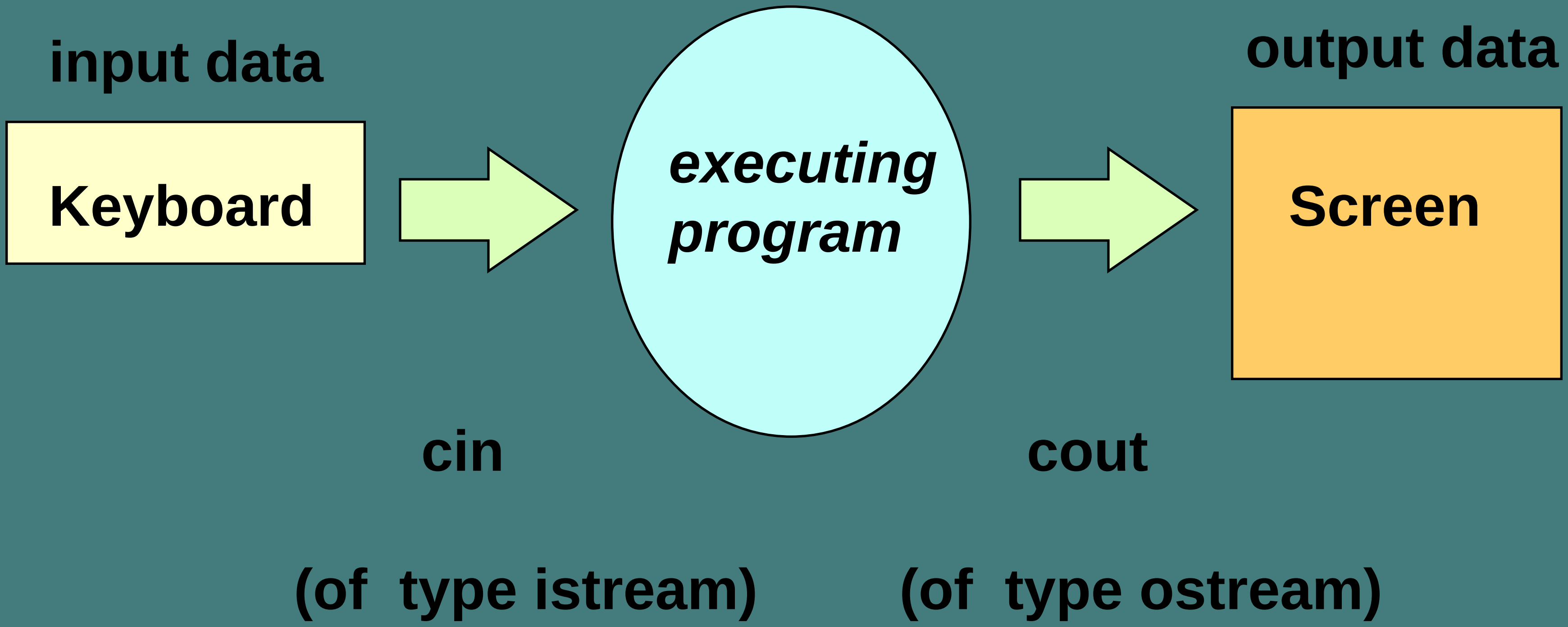
<iostream> is header file

for a library that defines 4 objects

- an istream object named **cin** (keyboard)
- an ostream object named **cout** (screen)
- an ostream object named **cerr** (screen)
- an ostream object named **clog** (screen)

Keyboard and Screen I/O

#include <iostream>



Chapter 13 Topics(part 1)

- Input/Output Stream in C++
  - Classes in I/O Library
  - Prompting for Interactive I/O
- Input Statements to Read Values for a Program
  - Using “>>”
  - Using Functions `get( )`
  - Using Functions `ignore( )`
  - Using Functions `getline( )`

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 ;            // prompt
cin    >>  quantity ;

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

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

Chapter 13 Topics(part 1)

- Input/Output Stream in C++
  - Classes in I/O Library
  - Prompting for Interactive I/O
- Input Statements to Read Values for a Program
  - Using “>>”
  - Using Functions `get( )`
  - Using Functions `ignore( )`
  - Using Functions `getline( )`

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 ;
```

Whitespace Characters Include . . .

- blanks
- tabs
- end-of-line (newline) characters

The newline character is created by hitting Enter or Return at the keyboard, or by using the manipulator endl or “\n” in a program.

Extraction Operator >>

“skips over”

(actually reads but does not store anywhere)

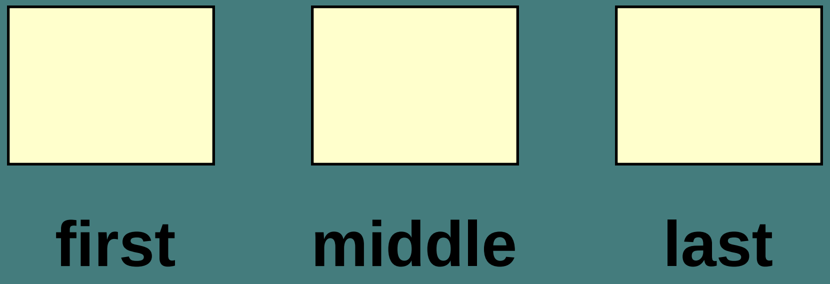
leading white space characters

as it reads your data from the input stream
(either keyboard or disk file)

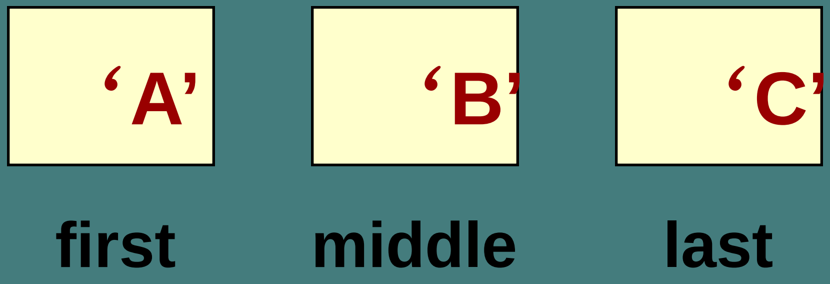
At keyboard you type:

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

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



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

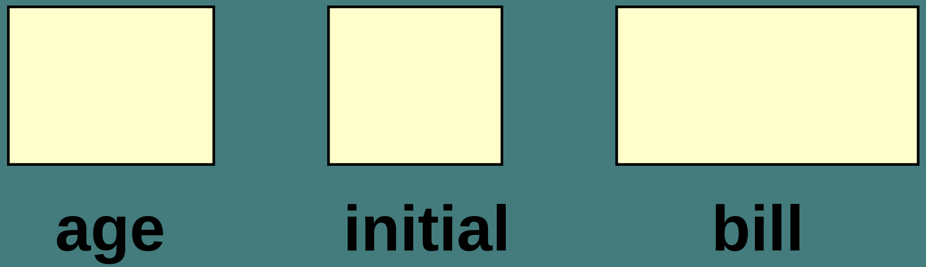


NOTE: A file reading marker is left pointing to the newline character after the 'C' in the input stream.

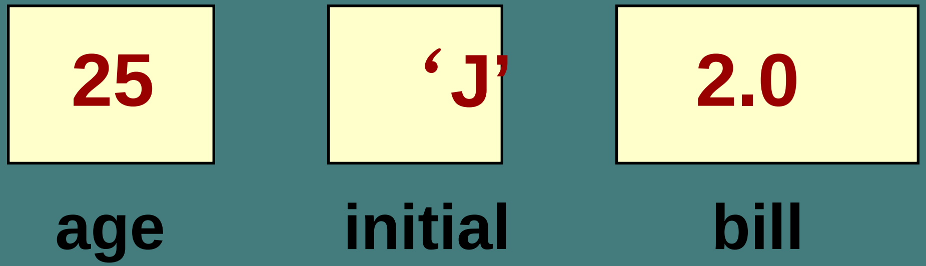
At keyboard you type:

`[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.

Another example using >>

NOTE: $\boxed{}$ shows the location of the file reading marker

STATEMENTS	CONTENTS			MARKER POSITION
	i	ch	x	
int i ; char ch ; float x ;				$\boxed{2}$5 A\n 16.9\n
cin >> i ;	25			25 $\boxed{}$A\n 16.9\n
cin >> ch ;	25	‘A’		25 A$\boxed{\backslash n}$ 16.9\n
cin >> x ;	25	‘A’	16.9	25 A\n 16.9$\boxed{\backslash n}$

Chapter 13 Topics(part 1)

- Input/Output Stream in C++
  - Classes in I/O Library
  - Prompting for Interactive I/O
- Input Statements to Read Values for a Program
  - Using “>>”
  - Using Functions `get( )`
  - Using Functions `ignore( )`
  - Using Functions `getline( )`

Another Way to Read `char` Data

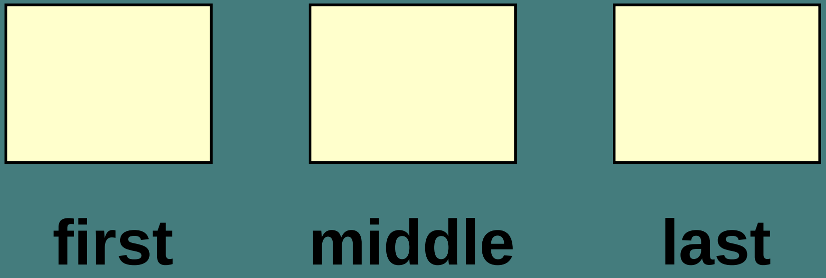
The `get( )` function can be used to read a single character.

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

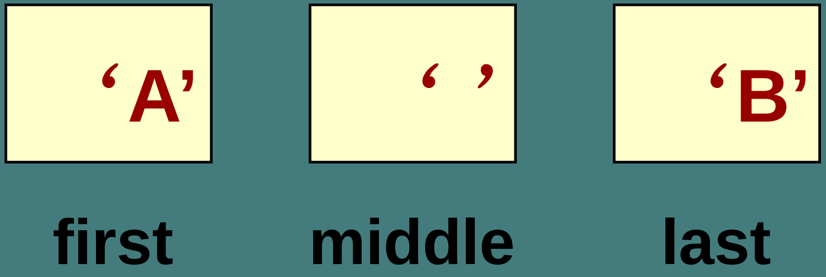
At keyboard you type:

`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.

Chapter 13 Topics(part 1)

- Input/Output Stream in C++
  - Classes in I/O Library
  - Prompting for Interactive I/O
- Input Statements to Read Values for a Program
  - Using “>>”
  - Using Functions `get( )`
  - Using Functions `ignore( )`
  - Using Functions `getline( )`

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
	a	b	c	
int a ; int b ; int c ;				□957 34 1235\n 128 96\n
cin >> a >> b ;	957	34		957 34□ 1235\n 128 96\n
cin.ignore(100, ‘\n’) ;	957	34		957 34 1235\n □128 96\n
cin >> c ;	957	34	128	957 34 1235\n 128□ 96\n

Another Example Using cin.ignore( )

NOTE: $\boxed{\ }$ shows the location of the file reading marker

STATEMENTS	CONTENTS i	CONTENTS ch	MARKER POSITION
`int i ;` `char ch ;`			$\boxed{\text{A}}$ 22 B 16 C 19\n
`cin >> ch ;`		'A'	A $\boxed{\ }$22 B 16 C 19\n
`cin.ignore(100, 'B') ;`		'A'	A 22 B $\boxed{\ }$16 C 19\n
`cin >> i ;`	16	'A'	A 22 B 16 $\boxed{\ }$C 19\n

Chapter 13 Topics(part 1)

- Input/Output Stream in C++
  - Classes in I/O Library
  - Prompting for Interactive I/O
- Input Statements to Read Values for a Program
  - Using “>>”
  - Using Functions `get( )`
  - Using Functions `ignore( )`
  - Using Functions `getline( )`

String Input in C++

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

EXAMPLE

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

HOWEVER . . .

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)

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 `getline` function with 2 arguments to overcome this obstacle（障碍）
- 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 );
```

“ Joe Hernandez 23”

firstName

?

lastName