



Program Input and the Software Design Process

Chapter 4 Topics

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

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)

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 ;
```



first

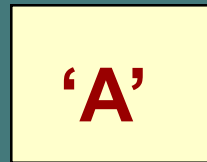


middle



last

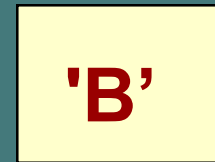
```
cin.get ( first ) ;  
cin.get ( middle ) ;  
cin.get ( last ) ;
```



first



middle



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

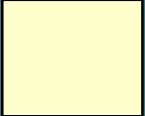
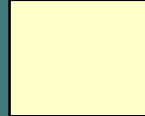
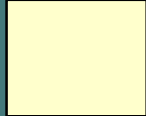


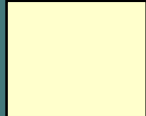

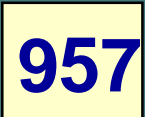



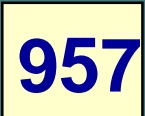

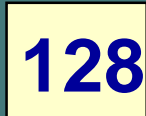

The **`ignore()`** function is used to skip (read and discard) characters in the input stream. The call

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

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




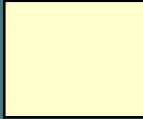


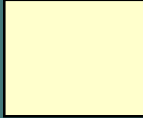



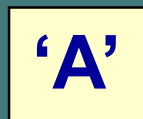

An Example Using cin.ignore()

NOTE:  shows the location of the file reading marker

STATEMENTS	CONTENTS			MARKER POSITION
<pre>int a ; int b ; int c ; cin >> a >> b ;</pre>				957 34 1235\n 128 96\n
	a	b	c	
				957 34  1235\n 128 96\n
	a	b	c	
<pre>cin.ignore(100, '\n') ;</pre>				957 34 1235\n  128 96\n
	a	b	c	
<pre>cin >> c ;</pre>				957 34 1235\n 128  96\n
	a	b	c	

Another Example Using cin.ignore()

NOTE:  shows the location of the file reading marker

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

String Input in C++

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

EXAMPLE

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

HOWEVER . . .

getline() Function

- ❖ Defined in the header file -- string
- ❖ 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 );
```

Joe Hernandez 23

firstName

?

lastName

Test_program

run_program

Test programs

```
//This program to count how many characters in a line,  
//skip the leading white blanks  
#include <iostream>  
#include <string>  
using namespace std ;  
void main()  
{ char ch ;  
  int count = 0 ; }  
  
cin >> ch ; //skip all leading white characters  
while( ch != '\n' ) // != 表示不等於  
{ cout << ch ; //逐字列印  
  count ++ ;  
  cin.get(ch) ; //逐字讀取包含其間空白  
}  
cout << "\nThe number of characters keyin :\n" ;  
cout << count << endl << '\n' ;  
}
```

宣告及設定初值

執行結果

Test programs

```
//This program to count how many characters in a line,  
//skip the leading white blanks  
#include<iostream>  
#include<string>  
using namespace std ;  
void main()  
{  char ch ;  
   int count = 0;  
   string line;  
  
   cin >> ch; //skip all white characters  
   getline(cin, line) ;  
   cout << line <<endl ;           //echo print少了第一個字母  
   cout << ch+line <<endl ;       //echo print  
   count = line.length() +1;  
   cout << "\nThe number of characters keyin :\n" ;  
   cout << count << endl ;  
}
```

執行結果

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 ;
```

```
#include <iostream>
using namespace std ;
int main()
{
    //initialize
    int n= 1 ; int sum = 0 ; int value ;
    //key in 10 numbers
    cout << "輸入10個整數，計算總和：" <<endl ;
    while (n <= 10 )
    {
        cin >> value ;
        cout << value <<endl ;    //echo print
        sum = sum + value ;    // sum+=value
        n = n+1 ;    // n++
    }
    cout << "the total sum : " << sum <<endl ;
    return 0 ;
}
```

這個程式輸入10個整數，並計算總和

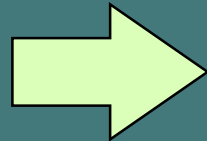
[Run program](#)

Diskette Files for I/O

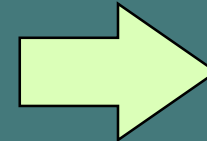
```
#include <fstream>
```

input data

disk file
"A:\myInfile.dat"



*executing
program*



output data

disk file
"A:\myOut.dat"

your variable

(of type ifstream)

your variable

(of type ofstream)

To Use Disk I/O, you must

- ❖ 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

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( );
```

雙斜線表示單
一文字'\'

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

```
#include <iostream>
#include <fstream>
using namespace std ;
int main()
{
    //declare and initialize
    int n= 1 ; int sum = 0 ; int value ;
    ifstream infile ; //宣告infile為輸入檔變數

    infile.open("myInfile.dat") ; //開啟資料檔
    infile >> value ;
    while ( infile ) //檔案尚未結束
    { cout << value << " " ; //echo print
      sum = sum + value ;
      infile >> value ;
    }
    cout << endl ; cout << "the total sum : " << sum << endl ;
    infile.close() ; return 0 ;
}
```

[run program](#)

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 OVERTIME = 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 for output
    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; //將計算結果寫入檔案

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

```
    cout << "Total payroll is " << total << endl;
    payFile.close() ;
    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;
}
```

程式執行結果

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

Test the state of an I/O Stream

```
#include <iostream>
#include <fstream>
using namespace std ;
int main()
{ int idNum ;
  float wages ;
  ifstream inFile ;
  inFile.open( "wages.dat" ) ;
  if ( !inFile ) //如果開啟不成功, !inFile 為true
  { cout << "can't open the input file. " ;
    return 1 ;
  }
```

Test the state of an I/O Stream

```
inFile >>idNUM >> wages ;  
//如果讀取資料成功, 則一直做到檔案結束  
while (inFile )  
{  
    cout << idNum << “ “ <<wages<<endl ;  
    inFile>> idNum >>wages ;  
}  
inFile.close() ;  
return 0 ;  
} //end of main()
```

程式執行結果

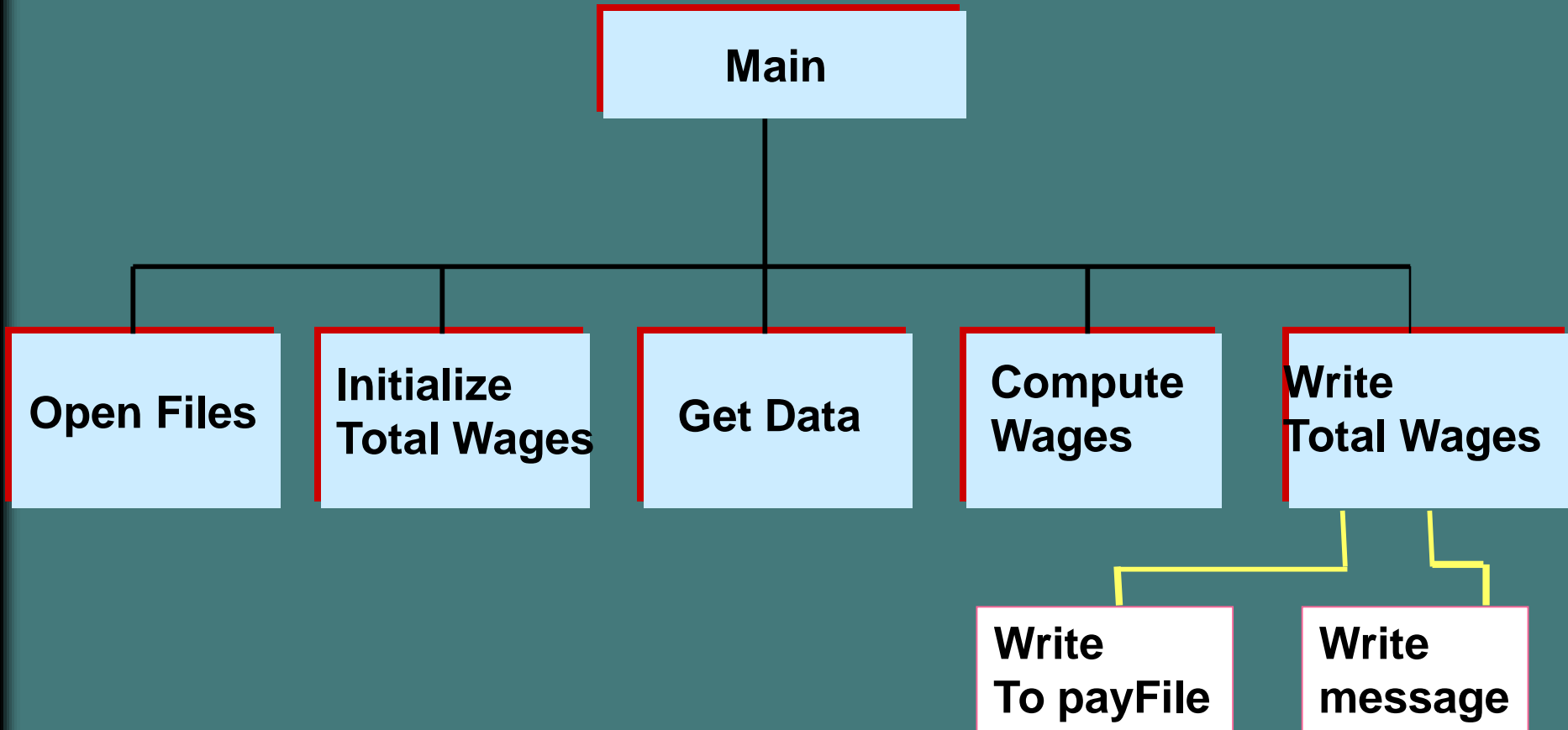
Entering File Name at Run Time

```
#include <string>           // contains conversion function c_str
#include <fstream>
ifstream  inFile ;
string    fileName;        //宣告fileName是string物件變數

cout << "Enter input file name : " << endl ;           // prompt
cin  >> fileName ;
inFile.open( fileName) // Compile-time error
                        // convert string fileName to a C string type

inFile.open( fileName.c_str( ) );
```

Module Structure Chart



Functional Decomposition

FOCUS is on actions and algorithms.

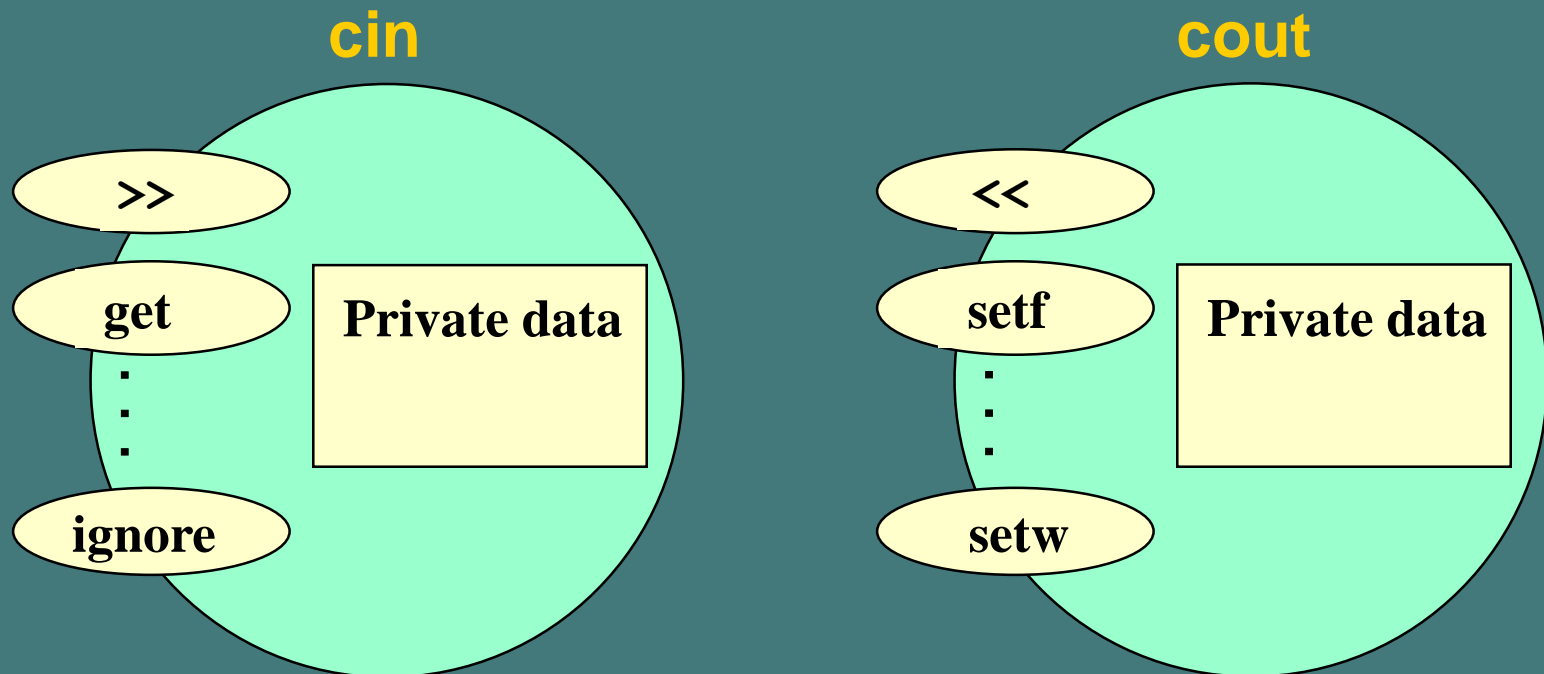
BEGINS by breaking the solution into a series of major steps. This process continues until each **subproblem** cannot be divided further or has an obvious solution.

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.

Object-Oriented Design(物件導向程式設計)

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, 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.

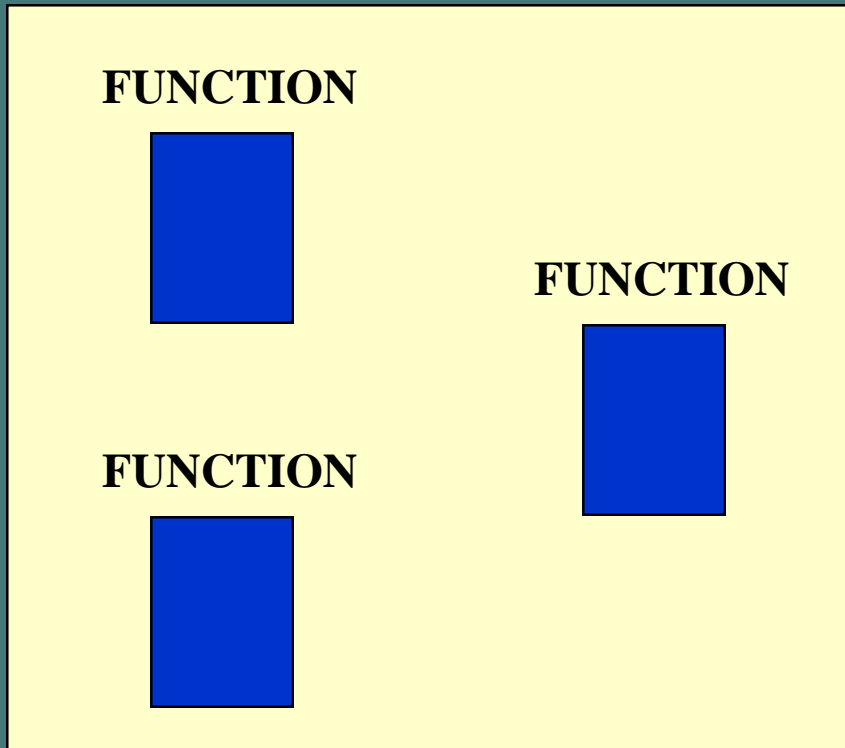
BEGINS by identifying the major objects in the problem, and choosing appropriate **operations** on those objects.

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

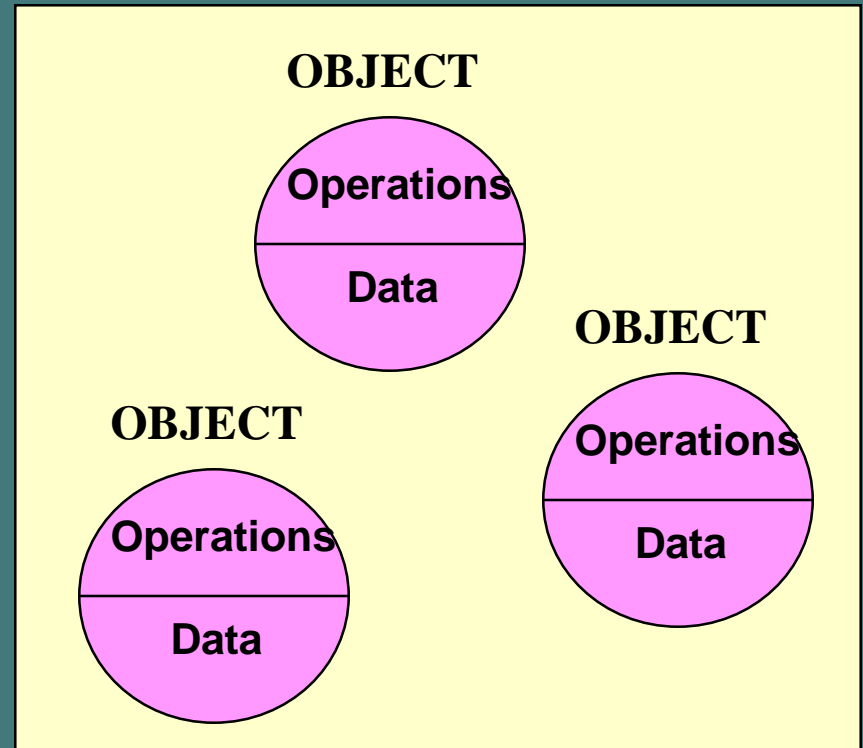
DATA plays a leading role. Algorithms are used to implement operations on the objects and to enable interaction of objects with each other.

Two Programming Methodologies

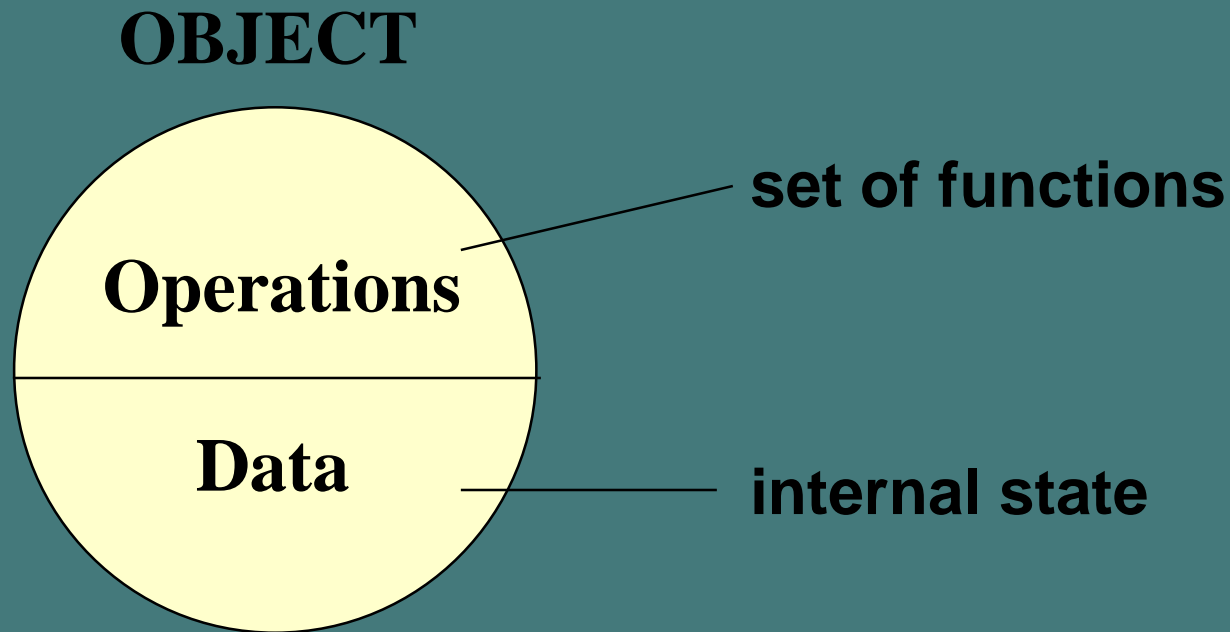
**Functional
Decomposition**



**Object-Oriented
Design**

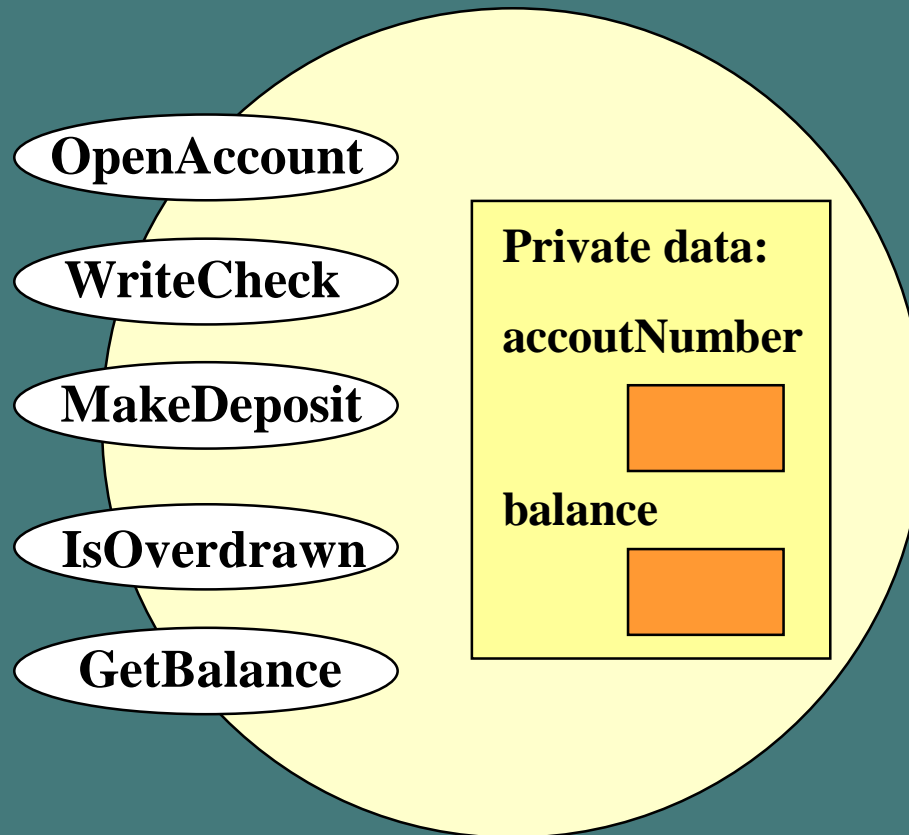


What is an object?



An object contains data and operations

checkingAccount



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