Part B - Foundations   
  
**Compound Types and Privacy**   
  
Workshop 2

In this workshop, you are to define a compound type with private data and public member functions.

**LEARNING OUTCOMES**

Upon successful completion of this workshop, you will have demonstrated the abilities

* to design a compound type
* to privatize data within a compound type
* to access data within an object of the compound type through public member functions
* to summarize what you have learned in the task

**Submission policy**

The “in-lab” section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (a 30% late deduction will be assessed). The “at-home” portion of the lab is **due the day before you next scheduled workshop.**

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly back up your work.

**Account Number Class – in-lab Section**

Get the lab files from the git repository (Github).You can use one of the following two ways:

1: On the lab computer or Matrix, issue this command to clone (download) the Workshop2 repository. (Select one of the two depending on your own preference)

**> git clone https://github.com/Seneca-244200/OOP-Workshop2.git**

**2: On a browser open this URL and click on Download Zip button to download the Workshop2 files in compressed zip format.**[**https://github.com/Seneca-244200/OOP-Workshop2.git**](https://github.com/Seneca-244200/OOP-Workshop2.git)

All the files needed for this workshop is already created and ready to use, if you are using windows platform on visual studio, just click on w2\_in\_lab.vcxproj to open the project.

Design and code a class named **AccountNumber,** in **AccountNumber.h** and **AccountNumber.cpp**.

Please note the compilation safeguards in the header file and the sict namespace. Starting from next workshop you must add these statements to your code.

**Adding predefined values to the project:**

Define the following values in AccountNumber.h

MAX\_NAME\_LENGTH 40

MIN\_BANKCODE 100

MAX\_BANKCODE 999

MIN\_BRANCHCODE 1

MAX\_BRANCHCODE 220

MIN\_ACCNO 10000

MAX\_ACCNO 99999

Create the AccountNumber Class that has four member variables named:

* **char \_name[MAX\_NAME\_LENGTH + 1];**
* **int \_bankCode;**
* **int \_branchCode;**
* **int \_accountNumber;**
* **bool \_validAccNumber;**

Ensure all of these are private.

Declare 4 public member functions named:

* **void name(const char name[])**
* **void accountNumber(int bankCode, int branchCode, int accountNumber)**
* **void display() const;**
* **bool isValid() const;**

The **name()** function sets the \_name character string in an **AccountNumber** object using the strcpy function.

The **accountNumber()** function sets the \_bankCode, \_branchCode and \_accountNumber integers in an **AccountNumber** object and sets \_validAccNumber to true or false depending on the value of \_bankCode, \_branchCode and \_accountNumber as follows:

For \_validAccNumber to be true the following three conditions should be true:  
bankCode should be >= MIN\_BANKCODE and <= MAX\_BANKCODE  
branchCode should be >= MIN\_BRANCHCODE and <= MAX\_BRANCHCODE  
accountNumber should be >= MIN\_ACCNO and <= MAX\_ACCNO.  
Otherwise the \_validAccNumber is set to false;

The **isValid()**function returns the value of \_validAccNumber

The **display()** function checks if the AccountNumber isValid(). If so, it displays the current **AccountNumber** object on standard output as follows:

**cout << "Name: " << \_name << ", Account number: " << \_bankCode << "-"**

**<< \_branchCode << "-" << \_accountNumber << endl;**

If AccountNumber is not Valid it displays:

**cout << \_name << " does not have a valid account number." << endl;**

The main program that uses your new class contains the following code.

// OOP244 Workshop 2: Compound types and privacy

// File w2\_in\_lab.cpp

// Version 1.0

// Date 2016/01/14

// Author Fardad Soleimanloo

// Description

// This file is used to demonstrate classes in C++ and

// how member variables can be defined as private but

// accessed through member functions

//

// Revision History

///////////////////////////////////////////////////////////

// Name Date Reason

//

///////////////////////////////////////////////////////////

#include <iostream>

using namespace std;

#include "AccountNumber.h"

using namespace sict;

int main(){

AccountNumber myNumber;

char name[41];

int bankCode;

int branchCode;

int accNumber;

cout << "Bank account app" << endl <<

"===================" << endl << endl;

cout << "Please enter your name: ";

cin >> name;

cout << "please enter your bank account ,branch code" <<

", and account number as follows:" << endl << "999 999 99999: ";

do{

cin >> bankCode >> branchCode >> accNumber;

myNumber.name(name);

myNumber.accountNumber(bankCode, branchCode, accNumber);

myNumber.display();

} while (!myNumber.isValid() &&

cout << "Invalid account number, (999 999 9999), try again: ");

cout << "Thank you!" << endl;

return 0;

}

Compiling and running the above code with your AccountNumber.cpp should “exactly” generate the following output:

Bank account app  
===================  
Please enter your name: John  
  
please enter your bank account ,branch code, and account number as follows:  
999 999 99999: 1 123 12345  
John does not have a valid account number.  
Invalid account number, (999 999 9999), try again: 1234 123 12345  
John does not have a valid account number.  
Invalid account number, (999 999 9999), try again: 123 0 12345  
John does not have a valid account number.  
Invalid account number, (999 999 9999), try again: 123 1234 12345  
John does not have a valid account number.  
Invalid account number, (999 999 9999), try again: 123 123 123  
John does not have a valid account number.  
Invalid account number, (999 999 9999), try again: 123 123 123456  
John does not have a valid account number.  
Invalid account number, (999 999 9999), try again: 123 123 12345  
Name: John, Account number: 123-123-12345  
Thank you!

**In-Lab SUBMISSION (80%)**

If not on matrix already, upload your **AccountNumber.h** and **AccountNumber.cpp** and **w2\_in\_lab.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

**~profname.proflastname/submit w2\_in\_lab <ENTER>**

Following the instructions, test and demonstrate execution of your program using the same data as the output example above.

**At Home Section: (20%)**

For the “At Home” Section of the workshop do the following:

1 - Create two private constant member functions called displayName and displayNumber. These two methods return void and have no arguments.

displayName, displays the name portion of the display() function only (no newline after):  
  
Name: John

displayNumber display the number portion of the display() function only (no newline after and no space or comma before):

Account number: 123-123-12345

2- Modify the display function of AccountNumber by adding two Boolean arguments; display\_name and display\_number.  
Using the two private display functions written in part 1 and default value for arguments re-implement the display function to work as follows:

**display() –** will provide the same output as before

**display(false) –** will only output the account number

**display(true, false) –** will only display the name

**display(false, false) –** will not output anything

The main program that uses your new implementation contains the following code.

// OOP244 Workshop 2: Compound types and privacy

// File w2\_at\_home.cpp

// Version 1.0

// Date 2016/01/14

// Author Fardad Soleimanloo

// Description

// This file is used to demonstrate classes in C++ and

// how member variables can be defined as private but

// accessed through member functions

//

// Revision History

///////////////////////////////////////////////////////////

// Name Date Reason

///////////////////////////////////////////////////////////

#include <iostream>

using namespace std;

#include "AccountNumber.h"

using namespace sict;

void displayAccountNumber(const AccountNumber\* acc);

int main(){

AccountNumber myNumber;

char name[41];

int bankCode;

int branchCode;

int accNumber;

cout << "Bank account app" << endl <<

"===================" << endl << endl;

cout << "Please enter your name: ";

cin >> name;

cout << "please enter your bank account ,branch code" <<

", and account number as follows:" << endl << "999 999 99999: ";

do{

cin >> bankCode >> branchCode >> accNumber;

myNumber.name(name);

myNumber.accountNumber(bankCode, branchCode, accNumber);

displayAccountNumber(&myNumber);

} while (!myNumber.isValid()

&& cout << "Invalid account number, (999 999 9999), try again: ");

cout << "Thank you!" << endl;

return 0;

}

void displayAccountNumber(const AccountNumber\* acc){

acc->display();

cout << "-------------" << endl;

acc->display(false);

cout << "-------------" << endl;

acc->display(true, false);

cout << "-------------" << endl;

acc->display(false, false);

}

Compiling and running the above code with your AccountNumber.cpp should “exactly” generate the following output:

**Bank account app**

**===================**

**Please enter your name: John**

**please enter your bank account ,branch code, and account number as follows:**

**999 999 99999: 1 123 12345**

**John does not have a valid account number.**

**-------------**

**John does not have a valid account number.**

**-------------**

**John does not have a valid account number.**

**-------------**

**John does not have a valid account number.**

**Invalid account number, (999 999 9999), try again: 123 123 12345**

**Name: John, Account number: 123-123-12345**

**-------------**

**Account number: 123-123-12345**

**-------------  
Name: John**

**-------------  
Thank you!**

**ReflectioN**

1. In a file called reflect.txt and using examples from your own code explain which features of object orientation you used.
2. In the main function, there is no “if” statement for printing an error message when “isValid()” returns false and yet the error message is printed when account is not valid, explain why.

**SUBMISSION**

If not on matrix already, upload your **AccountNumber.h** and **AccountNumber.cpp** , **w2\_at\_home.cpp and reflect.txt** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

**~profname.proflastname/submit w2\_in\_lab <ENTER>**

Following the instructions, test and demonstrate execution of your program using the same data as the output example above.