

# Compound Types and Privacy

## Workshop 3

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.

### CREDIT CARD 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 Workshop3 repository. (Select one of the two depending on your own preference)

```
> git clone https://github.com/Seneca-244200/OOP-Workshop3.git
```

2: On a browser open this URL and click on Download Zip button to download the Workshop3 files in compressed zip format.

<https://github.com/Seneca-244200/OOP-Workshop3>

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 `w3_in_lab.vcxproj` to open the project.

Design and code a class named `CreditCard`, in `CreditCard.h` and `CreditCard.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:

In the `CreditCard.h` file, you must define the following constants:

`MAX_NAME_LENGTH` with a value of 40. This value represents the maximum number of characters for the name of a cardholder.

`MIN_INST_NUMBER` with a value of 100. This is the lowest valid institution code.

`MAX_INST_NUMBER` with a value of 999. This is the highest valid institution code

`MIN_EXP_YEAR` with a value of 2017. The lowest valid value for the card's expiration year

`MAX_EXP_YEAR` with a value of 2037. The highest valid value for the card's expiration year

`MIN_CARD_NUMBER` with a value of 4000000000000000. The lowest valid value for the card number.

`MAX_CARD_NUMBER` with a value of 4000999999999999. The highest valid value for the card number.

Create the `CreditCard` Class with the following six members.

- `m_cardHolderName` of type `char[MAX_NAME_LENGTH]`
- `m_cardNumber` of type `unsigned long long`
- `m_institutionCode` of type `int`
- `m_expiryYear` of type `int`
- `m_expiryMonth` of type `int`
- `m_numberInTheBack` of type `int`

Ensure that only member functions of the class can access these data members.

The `CreditCard` class must have the following members:

- `void name(const char cardHolderName[])`
- `void initialize(unsigned long long creditCardNumber,`  
• `int instCode,`  
• `int expiryYear,`  
• `int expiryMonth,`  
• `int numberInTheBack)`
- `void write() const;`
- `bool isValid() const;`

The `name()` function copies the string from the parameter (`cardHolderName`) into the data member string (`m_cardHolderName`).

The `initialize()` function sets the `m_cardNumber`, `m_institutionCode`, `m_expiryYear`, `m_expiryMonth` and `m_numberInTheBack` data members with the information received from the parameters.

The `isValid()` function returns true if the information contained in the object represents a valid credit card. The function returns false otherwise. A credit card object is valid if:

- The cardholder name has at least one character.
- The credit card number is in the range of `MIN_CARD_NUMBER` and `MAX_CARD_NUMBER`
- The institution code is in the range of `MIN_INST_NUMBER` and `MAX_INST_NUMBER`
- The expiry year is in the range of `MIN_EXP_YEAR` and `MAX_EXP_YEAR`
- The expiry month is between 1 and 12
- The number in the back is positive and has no more than 3 digits.

```
Cardholder: Jane Doe
Card Number: 4999012398760001
Institution: 301
Expires: 10/2018
Number at the back: 505
```

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

```
// OOP244 Workshop 3: Compound types and privacy
// File      w3_in_lab.cpp
// Version 1.0
// Date      2017/01/15
// Author    Ed Arvelaez
// 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>
#include "CreditCard.h"
using namespace std;
using namespace sict;

int main() {
    CreditCard myCC;
    char name[41];
    int instCode;
    int expiryYear;
    int expiryMonth;
    unsigned long long cardNumber;
    int backNumber;
    char slash;

    cout << "Credit Card app" << endl <<
         "===== " << endl << endl;
    cout << "Please enter your name: ";
    cin >> name;

    do {
        cout <<
            "Please enter your credit card number, institution code, " <<
```

```

        "expiry date, and security number as follows: " <<
        "4000123412341234 999 12/1234 999" << endl << "> " ;

    cin >> cardNumber >> instCode >> expiryMonth >> slash >> expiryYear >>
        backNumber ;

    cout << endl;
    myCC.name(name);
    myCC.initialize(cardNumber, instCode, expiryYear, expiryMonth, backNumber);
    myCC.write();
} while (!myCC.isValid() && cout << "Invalid input" << endl );
cout << endl << "Thank you!" << endl;
return 0;

}

```

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

```

Credit Card app
=====

```

```

Please enter your name: John
Please enter your credit card number, institution code, expiry date, and security
number as follows: 4000123412341234 999 12/1234 999
> 1000111122223333 17 17/1972 15

```

```

Invalid input
Please enter your credit card number, institution code, expiry date, and security
number as follows: 4000123412341234 999 12/1234 999
> 4000111122223333 17 17/1972 15

```

```

Invalid input
Please enter your credit card number, institution code, expiry date, and security
number as follows: 4000123412341234 999 12/1234 999
> 4000111122223333 301 17/1972 15

```

```

Invalid input
Please enter your credit card number, institution code, expiry date, and security
number as follows: 4000123412341234 999 12/1234 999
> 4000111122223333 301 12/1972 15

```

```

Invalid input
Please enter your credit card number, institution code, expiry date, and security
number as follows: 4000123412341234 999 12/1234 999
> 4000111122223333 301 12/2020 15

```

```

Cardholder: John
Card Number: 4000111122223333
Institution: 301
Expires: 12/2020
Number at the back: 15

```

Thank you!

## IN-LAB SUBMISSION (50%)

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload your `CreditCard.h`, `CreditCard.cpp` and `w3_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 244_w3_lab <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

## AT HOME SECTION: (40%)

For the “At Home” Section of the workshop copy the CreditCard Module (`CreditCard.h` and `CreditCard.cpp`) to your at-home directory and do the following:

1 - Create two private constant member functions called `writeName` and `writeCardInfo`. These two methods return void and have no arguments.

`writeName`, displays the name portion of the `write()` function only (no newline after):

CardHolder: John

`writeCardInfo` displays the number portion of the `write()` function only (no newline after and no space or comma before):

Card Number: 4000111122223333  
Institution: 301  
Expires: 12/2020  
Number at the back: 15

2- Modify the write function of CreditCard by adding two Boolean arguments; displayName and displayCardInfo.

Using the two private write functions written in part 1 and default value for arguments re-implement the write function to work as follows:

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

**write(false)** - will only output the card information

**write(true, false)** - will only display the name

`write(false, false)` - will not output anything

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

```
// OOP244 Workshop 3: Compound types and privacy
// File      w3_at_home.cpp
// Version 1.0
// Date      2017/01/15
// Author     Ed Arvelaez
// 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 "CreditCard.h"
using namespace sict;

void writeAll(const CreditCard& );

int main() {
    CreditCard myCC;
    char name[41];
    int instCode;
    int expiryYear;
```

```

int expiryMonth;
unsigned long long cardNumber;
int backNumber;
char slash;

cout << "Credit Card app" << endl <<
    "=====" << endl << endl;
cout << "Please enter your name: ";
cin >> name;

do {
    cout <<
        "Please enter your credit card number, institution code, " <<
        "expiry date, and security number as follows: " <<
        "4000123412341234 999 12/1234 999" << endl << "> ";

    cin >> cardNumber >> instCode >> expiryMonth >> slash >> expiryYear >>
        backNumber;

    cout << endl;
    myCC.name(name);
    myCC.initialize(cardNumber, instCode, expiryYear, expiryMonth, backNumber);
} while (!myCC.isValid() && cout << "Invalid input" << endl);
cout << endl << "Thank you!" << endl;
writeAll(myCC);
return 0;
}

void writeAll(const CreditCard& card)
{
    card.write();
    cout << endl << "-----" << endl;
    card.write(false);
    cout << endl << "-----" << endl;
    card.write(true, false);
    cout << endl << "-----" << endl;
    card.write(false, false);
}

```

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

#### Credit Card app

=====

```

Please enter your name: John
Please enter your credit card number, institution code, expiry date, and
security number as follows: 4000123412341234 999 12/1234 999
> 1111222233334444 301 12/2020 505

Invalid input
Please enter your credit card number, institution code, expiry date, and
security number as follows: 4000123412341234 999 12/1234 999
> 4000111122223333 301 12/2020 505

```



Thank you!

Cardholder: John

Card Number: 4000111122223333

Institution: 301

Expires: 12/2020

Number at the back: 505

-----

Card Number: 4000111122223333

Institution: 301

Expires: 12/2020

Number at the back: 505

-----

Cardholder: John

-----

## REFLECTION (10%)

- 1- In a file called reflect.txt and using examples from your own code explain which features of object orientation you used.
- 2- Explain your understanding of the do-while loop written in the main program, especially the condition that makes it stop. Also explain the choice of type for storing the credit card number.

## AT-HOME SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload your `CreditCard.h` and `CreditCard.cpp` , `w3_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 244_w3_home <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.