

LOG1000 - TP4

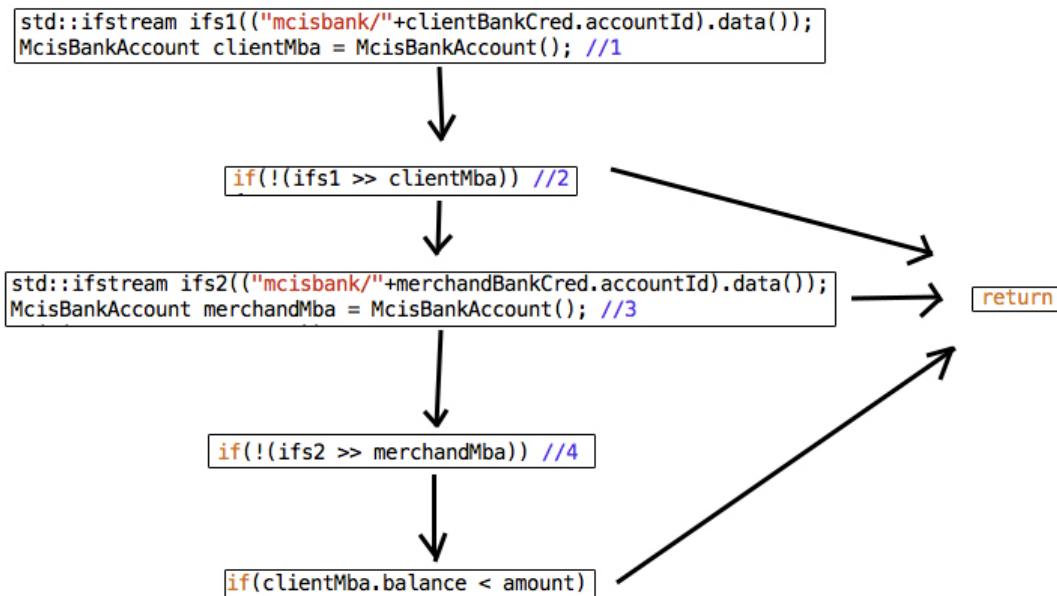
Samuel Rondeau & Pacôme Bondet de la Bernardie

E1.A

Les trois fonctions suivantes sont traduites en diagramme de flot de contrôle:

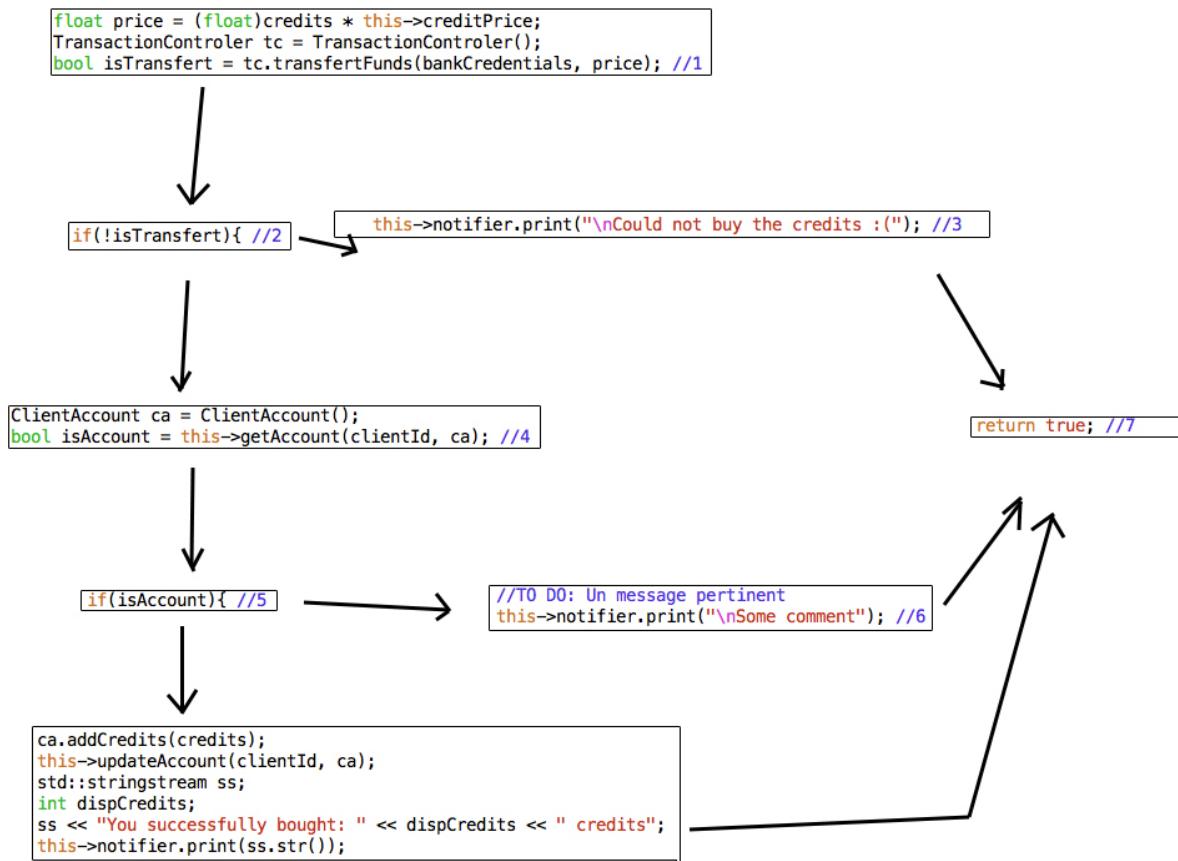
-dans McisBankTC.cpp

bool McisBankTC::transfertFunds(BankCredentials& merchantBankCred, BankCredentials& clientBankCred, float amount)

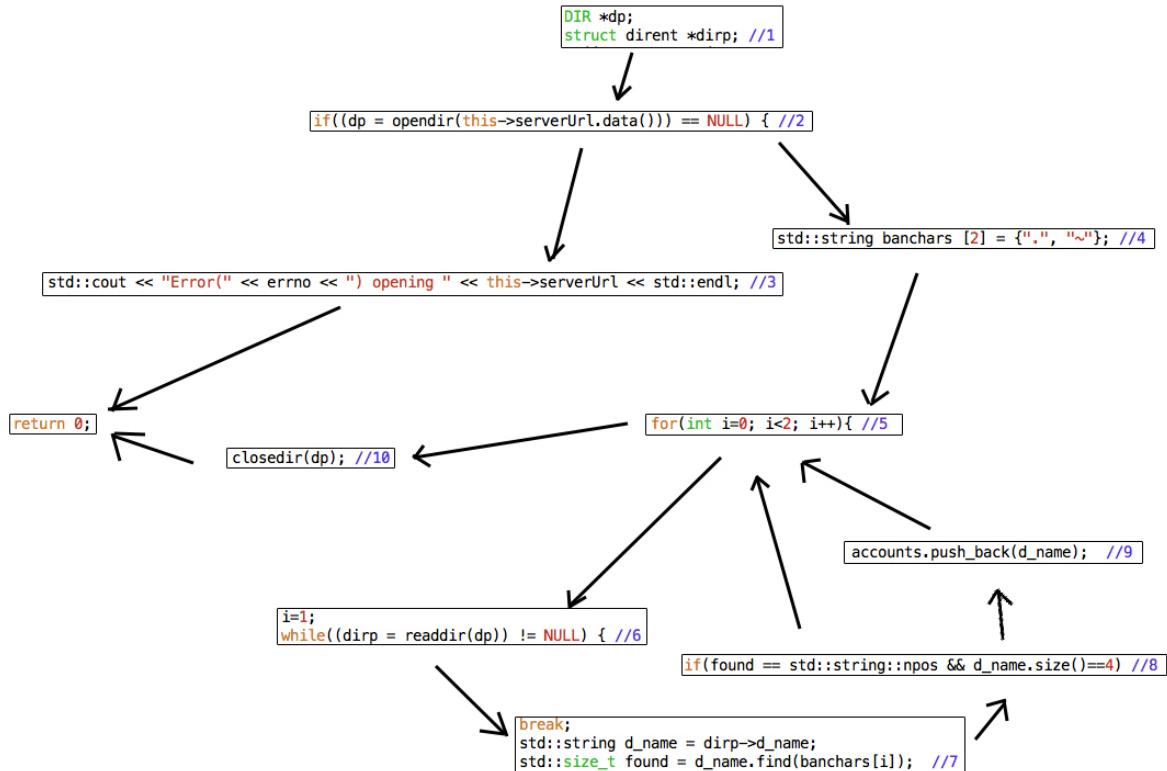


-dans OpusController.cpp

bool OpusController::addCredits(BankCredentials& bankCredentials, std::string clientId, unsigned int credits)

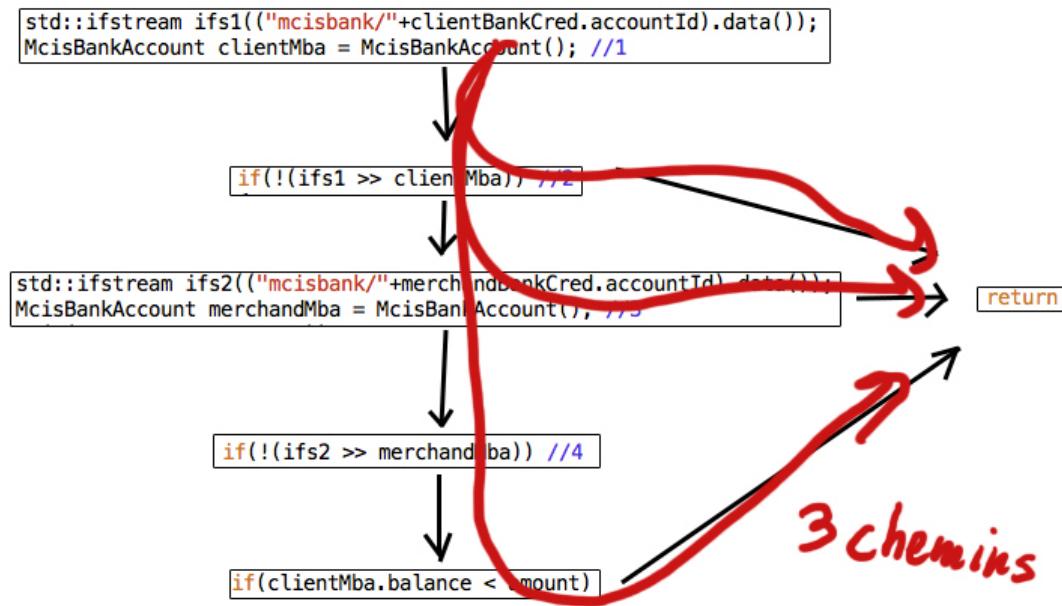


```
int OpusController::listAccounts(std::vector<std::string>& accounts)
```

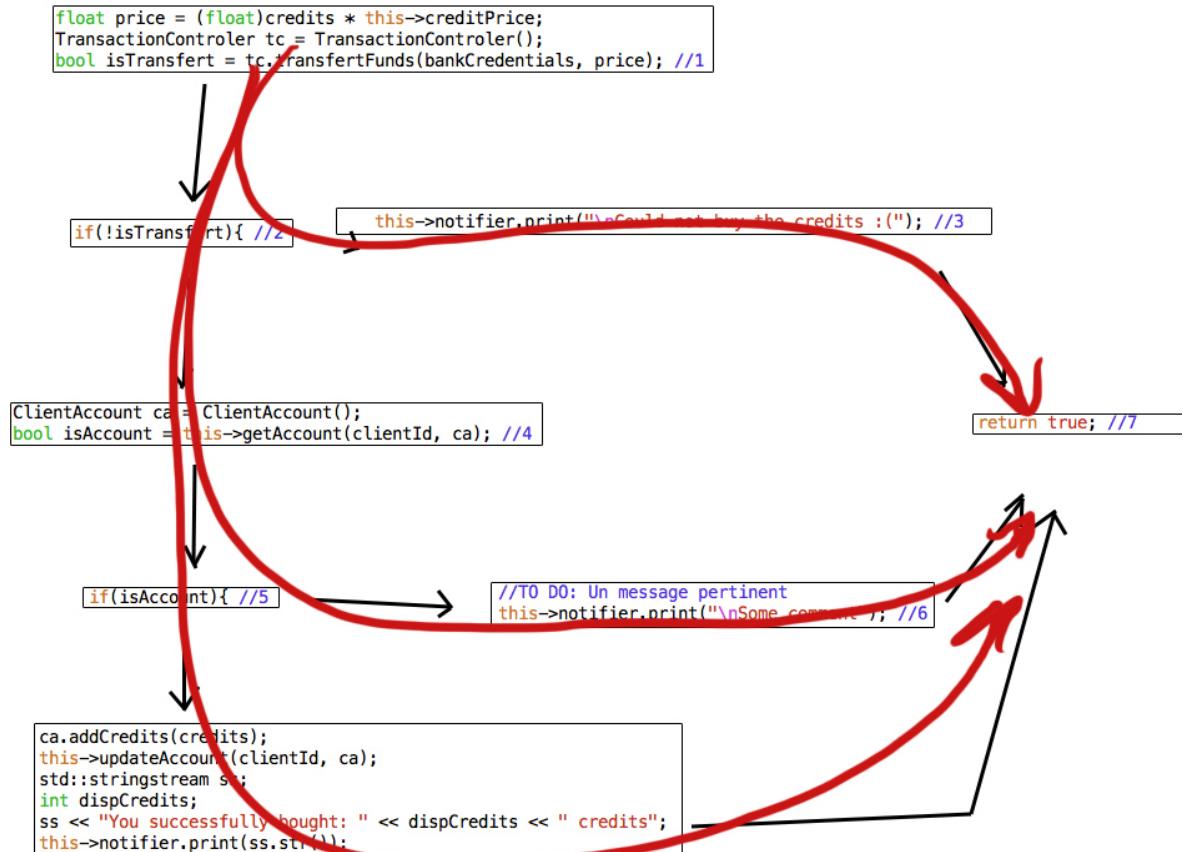


E1.B

```
bool McisBankTC::transfertFunds(BankCredentials& merchantBankCred, BankCredentials& clientBankCred, float amount)
```

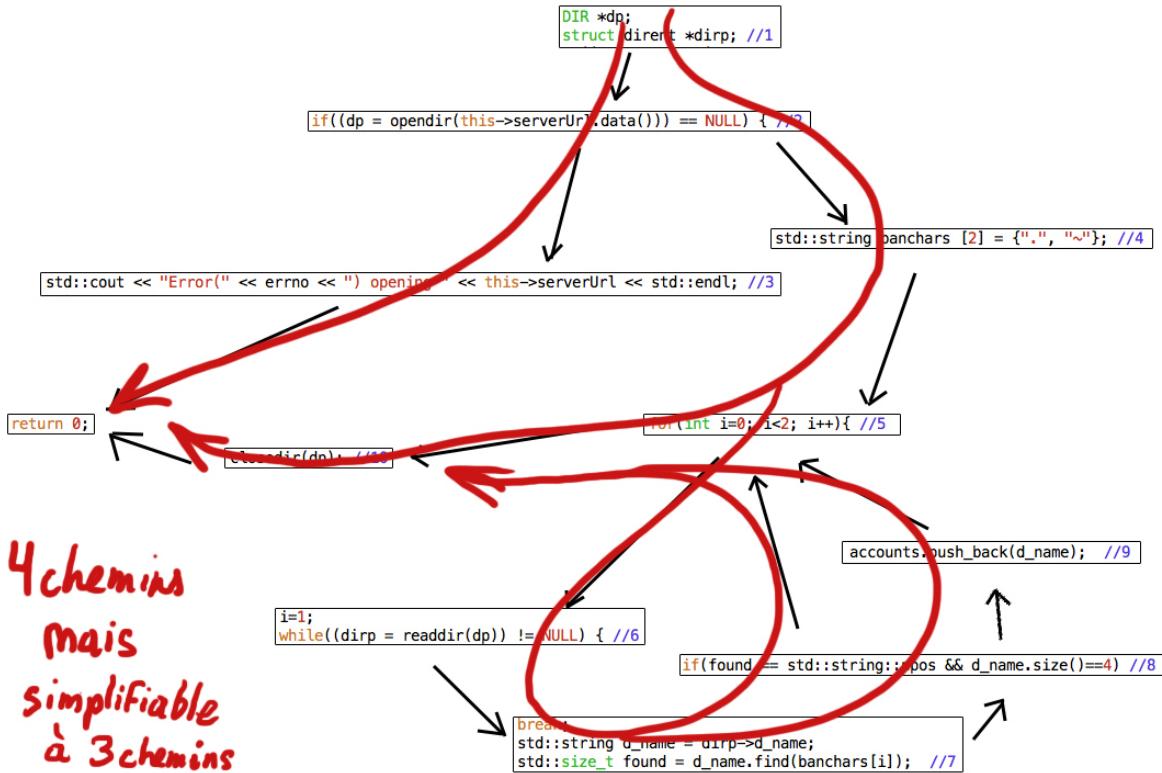


```
bool OpusController::addCredits(BankCredentials& bankCredentials, std::string clientId,  
unsigned int credits)
```



3chemistry

```
int OpusController::listAccounts(std::vector<std::string>& accounts)
```



E1.C

Pour les deux premiers Tests (les fonctions TransferFunds et AddCredits)
Leur complexité cyclomatique est de 3, c'est à dire c'est le nombre de tests maximaux à faire pour couvrir toutes les conditions.

Pour la dernière (listAccounts) la complexité cyclomatique est de 4. Mais on peut réduire le nombre de tests à 3 à cause de la boucle (voir dessin ci-dessus) pour couvrir les conditions à l'intérieur de la boucle.

E2.A

Ordre d'appels:

main

```
    OpusController::addCredits
        ClientAccount::addCredits
            Notifier::print
                TransactionController::transfertFunds
    OpusController::listAccounts
```

E2.B

```
void ClientAccount::addCredits(unsigned int credits)
this->credits = 0;
```

Entrée	Sortie
credits = 42	this->credits += 0
credits = 0	this->credits += 0

```
void Notifier::print(std::String str)
this->std::string printUrl = "test";
```

Entrée	Sorties
str = ""	Notifier: ofs <- testnotifier_out
str = "chaine"	Notifier: chaine ofs <- testnotifier_outchaine

```
bool McisBankTC::transferFunds(BankCredentials& merchantBankCred, BankCredentials&
clientBankCred, float amount)
```

Entrées	Sortie
clientBankCred merchantBankCred do not exist	false
clientBankCred && merchantBankCred exist	

clientBankCred.balance < amount	false
clientBankCred.balance >= amount	true

```
int OpusController::listAccounts(std::vector<std::string>& accounts)
this->serverUrl;
```

Entrée	Sortie
File serverUrl is NULL or does not exist	Show error message return error number
else	return 0

```
bool OpusController::addCredits(BankCredentials& bankCredentials, std::string clientId,
unsigned int credits)
this->notifier;
this->creditPrice = 42;
```

Entrées	Sorties
bankCredentials is invalid in tc	false
clientId does not exist via getAccount()	false
balance < amount	false
bankCredentials is valid AND clientId exists AND balance >= amount	true

```
bool TransactionController::transfertFunds(BankCredentials& bc, float amount)
this->mcisBankTC;
```

Entrées	Sorties
File bc does not exist	false
clientMba.balance < amount	false
clientMba.balance >= amount	true

E3.b)

```
TestNotifier::testPrint : OK
OK (1)
TestClientAccount::testAddCredits : assertion
TestClientAccount.cpp:31:Assertion
Test name: TestClientAccount::testAddCredits
assertion failed
- Expression: clientAccount.credits == 4
```

Failures !!!

```
Run: 1 Failure total: 1 Failures: 1 Errors: 0
TestOpusController::testAddCreditsNoBankCredentials : OK
TestOpusController::testAddCreditsNoClientID : OK
TestOpusController::testAddCredits : OK
TestOpusController::testAddCreditsNotEnough : OK
TestOpusController::testListAccountsError(2) opening ../accounts/
: assertion
TestOpusController::testListAccountsNotExistFileError(2) opening notaccounts/
: OK
TestOpusController.cpp:82:Assertion
Test name: TestOpusController::testListAccounts
assertion failed
- Expression: result == 0
```

Failures !!!

```
Run: 6 Failure total: 1 Failures: 1 Errors: 0
TransactionControllerTest::testTransfertFunds : OK
TransactionControllerTest::testTransfertFundsBalanceSmall : OK
TransactionControllerTest::testTransfertFundsBalanceGood : OK
OK (3)
```

E3.c)

Les tests qui font défaut sont testClientAccount::testAddCredits et testNotifier::testPrint. testAddCredits, qui devrait ajouter x crédits, or le client ne reçoit pas x crédits. testPrint détermine que la chaîne de caractères enregistrée en fichier n'est pas égale au résultat attendu (il y a un 'l' de trop à la fin). Pour testAddCredits, toute valeur non-nulle cause défaut. Toute entrée cause défaut pour testPrint.

E4.A)

-Notifier::print

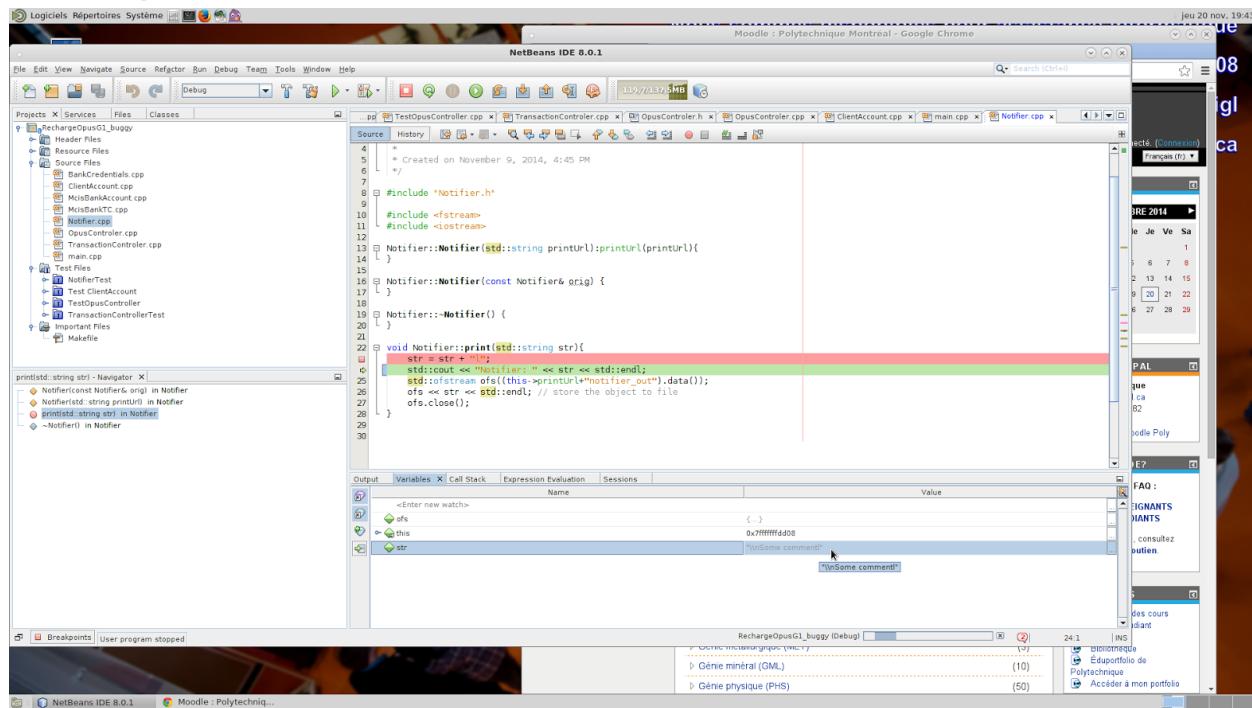


Figure 4.1 - Test de Notifier::print()

On remarque clairement la ligne que `str = str + "l"` est un problème.

Méthodologie utilisée: regarder et effacer ce problème très clairement évident, impossible à manquer, à se demander si c'est possible que ce "bug" puisse être là autrement que par exprès.

-OpusController::addCredits

On voudrait entrer dans la condition résultant `return false` si `isAccount` est faux, or on y entre lorsque `isAccount` est vrai. Le débuggeur le démontre. Solution: inverser la condition.

```

bool OpusController::addCredits(BankCredentials& bankCredentials, std::string clientId, unsigned int credits) {
    float price = (float)credits * this->creditPrice;
    TransactionController tc = TransactionController();
    bool isTransfer = tc.transferFunds(bankCredentials, price);
    if(!isTransfer) {
        this->notifier.print("I could not buy the credits :(");
        return false;
    }
    ClientAccount ca = ClientAccount();
    bool isAccount = this->getAccount(clientId, ca);
    if(isAccount) {
        //TO DO: Un message pertinent
        this->notifier.print("\nSome comment");
        return false;
    }
    ca.addCredits(credits);
    this->updateAccount(clientId, ca);
    std::stringstream ss;
    int dispCredits;
    ss << "You successfully bought: " << dispCredits << " credits";
    this->notifier.print(ss.str());
    return true;
}

```

Figure 4.2 - Test de OpusController::addCredits

```

bool OpusController::addCredits(BankCredentials& bankCredentials, std::string clientId, unsigned int credits) {
    float price = (float)credits * this->creditPrice;
    TransactionController tc = TransactionController();
    bool isTransfer = tc.transferFunds(bankCredentials, price);
    if(!isTransfer) {
        this->notifier.print("I could not buy the credits :(");
        return false;
    }
    ClientAccount ca = ClientAccount();
    bool isAccount = this->getAccount(clientId, ca);
    if(isAccount) {
        //TO DO: Un message pertinent
        this->notifier.print("\nError: No account Found");
        return false;
    }
    ca.addCredits(credits);
    this->updateAccount(clientId, ca);
    std::stringstream ss;
    int dispCredits;
    ss << "You successfully bought: " << dispCredits << " credits";
    this->notifier.print(ss.str());
    return true;
}

```

Figure 4.3 - Test de OpusController::addCredits

On trouve un autre problème: on affiche qu'on a acheté 32767 crédits, mais la valeur affichée n'est pas le bon nombre de crédits. La valeur affichée est celle de dispCredits, un entier non-initialisé qui prend alors la valeur maximale d'un entier signé, 32767. Retirons-le et affichons credits.

-ClientAccount::addCredits

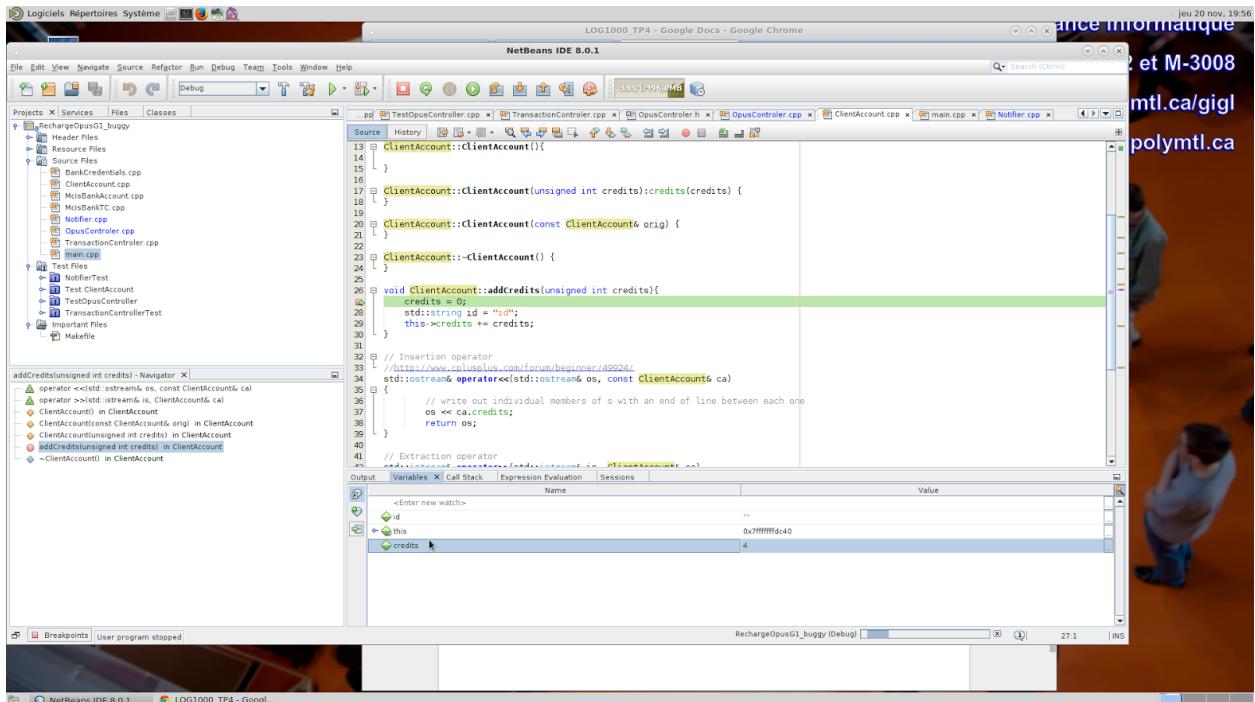


Figure 4.4 - Test de ClientAccount::addCredits

Ensuite, on voit encore très, très, très clairement la ligne causant l'erreur dans addCredits. La ligne credits = 0; ne devrait définitivement pas être là. Le debuggeur démontre que credits = 4 avant qu'il soit remis inutilement à 0. Cependant, le debuggeur a aussi montré un phénomène intéressant: après l'exécution de credits = 0, credits n'égalait pas 0, mais encore 4. On pourrait donc s'attendre que this->credits soit donc à 9 (car $5+4=9$), mais pourtant, this->credits vaut toujours 5, donc credits vaut 0.

Enlevons complètement credits = 0; et la méthode fonctionne bien.

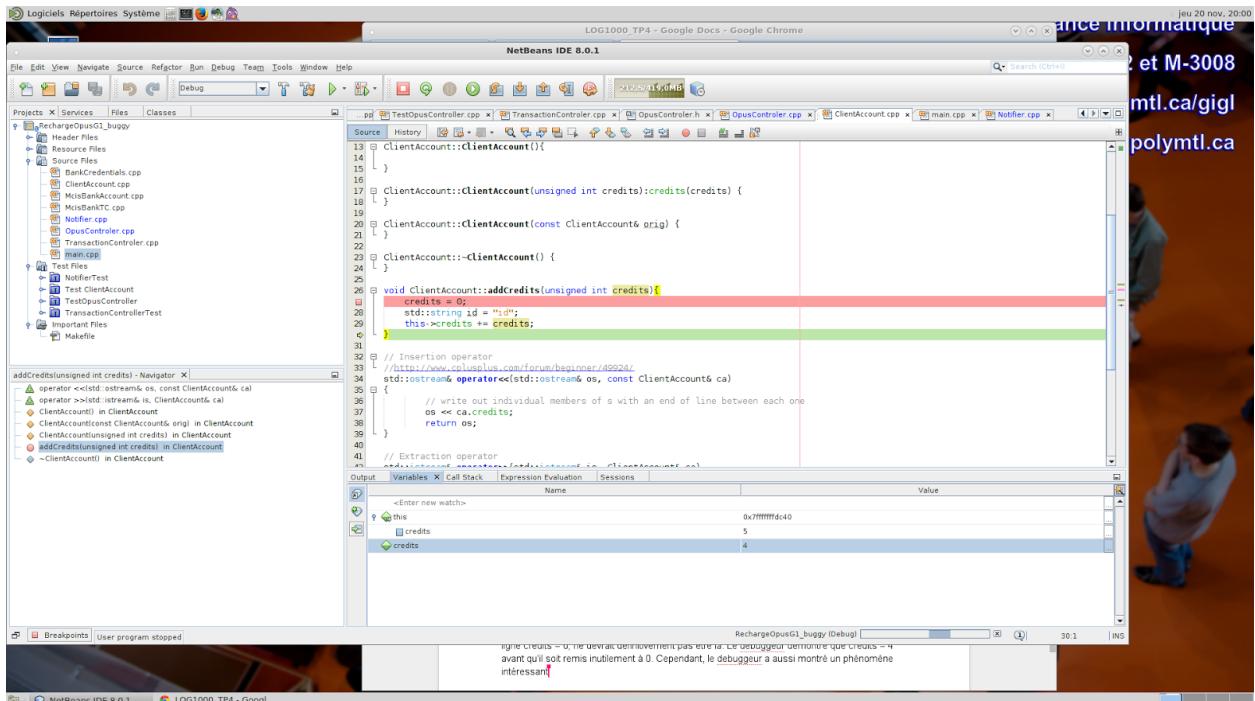


Figure 4.5 - Test de ClientAccount::addCredits

-OpusController::listAccounts

Testons listAccounts. On y retrouve du code bien étrange. Premièrement, une boucle for qui ne s'exécute nécessairement qu'une seule fois, et un break immédiatement à l'entrée d'une boucle while, ce qui n'est ni sensé être là, ni une bonne pratique de programmation. Par le debuggeur (voir les figures 4.6, 4.7 et 4.8 ci-bas), on remarque effectivement que l'exécution de la boucle for n'est faite qu'une fois. Solution: retirer la ligne `i = 1`.

Étrangement, la condition pour entrer dans la boucle while est vraie, mais on n'y rentre pas. Le pointeur retournée par l'affectation `dirp = readdir(dp) = 0x606130` est effectivement non-nulle. Retirons `break;`. Maintenant, l'exécution entre dans la boucle while.

```

48     int dispCredits;
49     ss << "You successfully bought: " << dispCredits << " credits";
50     this->notifier.print(ss.str());
51     return true;
52 }
53
54 @ int OpusController::listAccounts(std::vector<std::string>& accounts){
55     DIR *dp;
56     struct dirent *dptr;
57     if((dp = opendir(this->serverUrl.data())) == NULL) {
58         std::cout << "Error(" << errno << ") opening " << this->serverUrl << std::endl;
59         return errno;
60     }
61     std::string banchars [2] = {"-", "\0"};
62     for(int i=0; i<2; i++){
63         if((dptr = readdir(dp)) != NULL) {
64             i++;
65             while(dptr = readdir(dp)) != NULL) {
66                 break;
67             }
68             std::string d_name = dptr->d_name;
69             std::size_t found = d_name.find(banchars[i]);
70             if(found != std::string::npos && d_name.size() == 4)
71                 accounts.push_back(d_name);
72         }
73     }
74     closedir(dp);
75     return 0;
76 }

```

Figure 4.6 - Test de OpusController::listAccounts

```

48     int dispCredits;
49     ss << "You successfully bought: " << dispCredits << " credits";
50     this->notifier.print(ss.str());
51     return true;
52 }
53
54 @ int OpusController::listAccounts(std::vector<std::string>& accounts){
55     DIR *dp;
56     struct dirent *dptr;
57     if((dp = opendir(this->serverUrl.data())) == NULL) {
58         std::cout << "Error(" << errno << ") opening " << this->serverUrl << std::endl;
59         return errno;
60     }
61     std::string banchars [2] = {"-", "\0"};
62     for(int i=0; i<2; i++){
63         if((dptr = readdir(dp)) != NULL) {
64             i++;
65             while(dptr = readdir(dp)) != NULL) {
66                 break;
67             }
68             std::string d_name = dptr->d_name;
69             std::size_t found = d_name.find(banchars[i]);
70             if(found != std::string::npos && d_name.size() == 4)
71                 accounts.push_back(d_name);
72         }
73     }
74     closedir(dp);
75     return 0;
76 }
77 @ void OpusController::updateAccount(std::string id, ClientAccount& ca){
78 }

```

Figure 4.7 - Test de OpusController::listAccounts

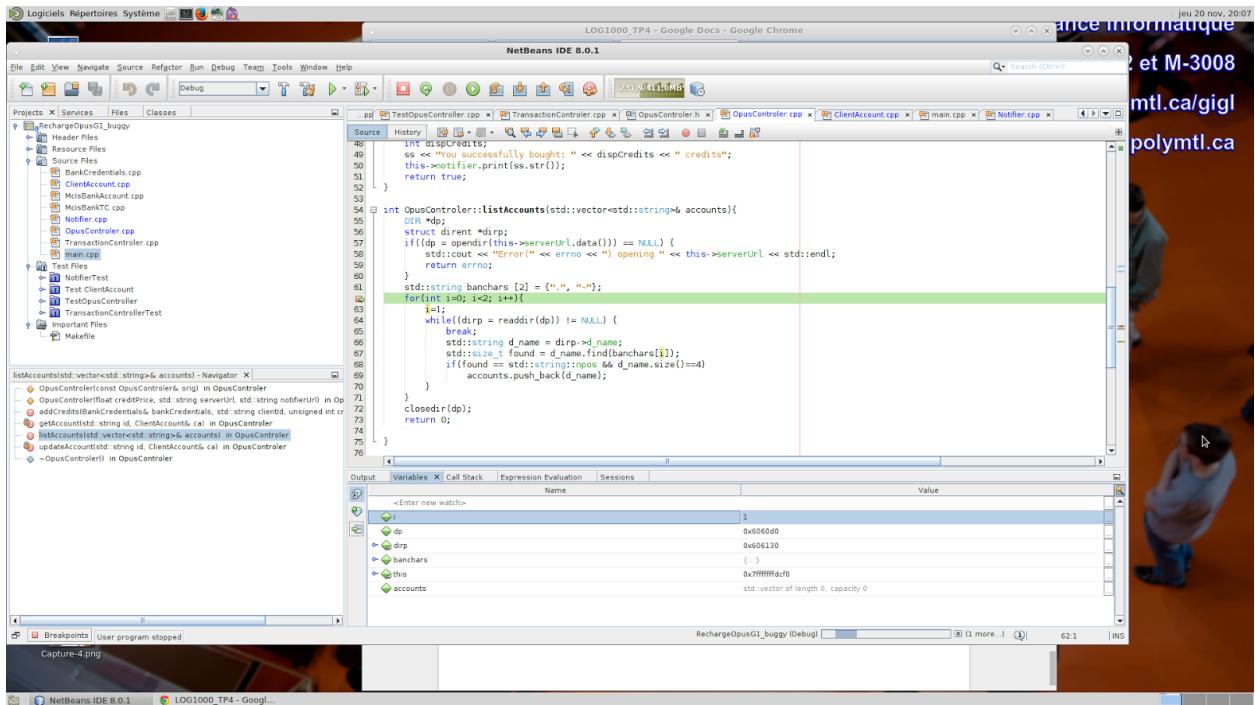


Figure 4.8 - Test de OpusController::listAccounts

E4.C)

TestNotifier::testPrint : OK

OK (1)

TestClientAccount::testAddCredits : OK

OK (1)

TestOpusController::testAddCreditsNoBankCredentials : OK

TestOpusController::testAddCreditsNoClientID : OK

TestOpusController::testAddCredits : OK

TestOpusController::testAddCreditsNotEnough : OK

TestOpusController::testListAccounts : OK

TestOpusController::testListAccountsNotExistFileError(2) opening notaccounts/

: OK

OK (6)

TransactionControllerTest::testTransfertFunds : OK

TransactionControllerTest::testTransfertFundsBalanceSmall : OK

TransactionControllerTest::testTransfertFundsBalanceGood : OK

OK (3)

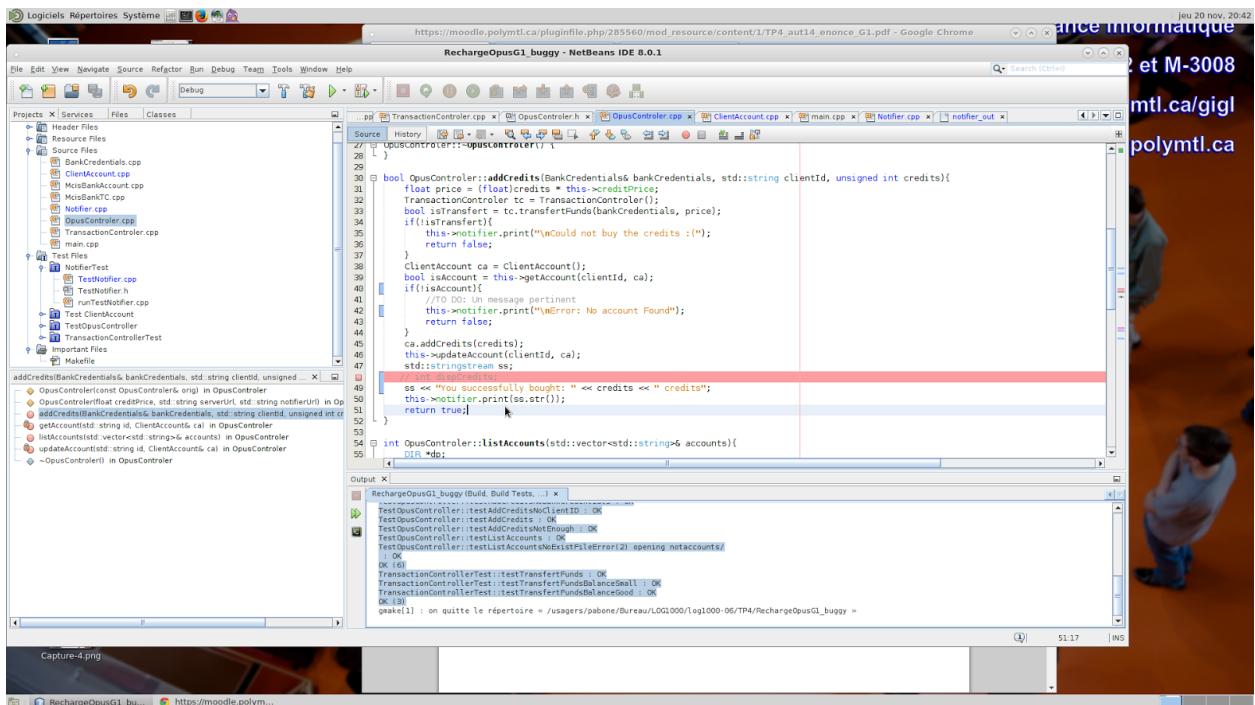
Tous les tests réussissent.

E5.A

Le “Data Flow Testing” est similaire aux tests de flots de contrôle mais à la place de regarder seulement le flot du programme on s’interesse aussi plus précisement aux variables: leur affectation, allocation, désallocation, utilisation etc..

On va chercher en plus que d’autres tests le détail sur l’utilisation de chaque variable pour voir si elles sont utilisé adéquatement. (On cherchera par exemple des variables qui seraient utilisées avant même d’être déclarés.)

E5.B



The screenshot shows the NetBeans IDE interface. The code editor displays the `OpusController.cpp` file, specifically the `addCredits` method. The output window shows the results of a build and test process for the `RechargeOpusGL_buggy` project, indicating various test cases passed or failed.

```
OpusController::addCredits(BankCredentials& bankCredentials, std::string clientId, unsigned int credits)
{
    float price = (float)credits * this->creditsPrice;
    TransactionController tc = TransactionController();
    bool isAccount = tc.transferFunds(bankCredentials, price);
    if(!isTransfer){
        this->notifier.print("\nCould not buy the credits :(");
        return false;
    }
    ClientAccount ca = ClientAccount();
    bool isAccount = this->getAccount(clientId, ca);
    if(!isAccount){
        //TO DO: Un message pertinent
        this->notifier.print("Error: No account Found");
        return false;
    }
    ca.addCredits(credits);
    this->updateAccount(clientId, ca);
    std::stringstream ss;
    ss << "You successfully bought: " << credits << " credits";
    this->notifier.print(ss.str());
    return true;
}

int OpusController::listAccounts(std::vector<std::string>& accounts)
{
    DIR *dir;
    struct dirent *dirent;
    std::string path = "/home/polymtl/Bureau/LOG1000/og1000-06/TP4/RechargeOpusGL_buggy";
    if((dir = opendir(path.c_str())) != NULL)
    {
        while((dirent = readdir(dir)) != NULL)
        {
            if(strcmp(dirent->d_name, ".") != 0 && strcmp(dirent->d_name, "..") != 0)
                accounts.push_back(dirent->d_name);
        }
        closedir(dir);
    }
}
```

Output window content:

```
RechargeOpusGL_buggy [Build Test...]
TestOpusController::testAddCreditsWithClientID : OK
TestOpusController::testAddCreditsNotEnough : OK
TestOpusController::testListAccounts : OK
TestOpusController::testListAccountNameWithFileError() opening notaccounts/ : OK
TestOpusController::testTransferFunds_OK
TransactionControllerTest::testTransferFundsBalanceSmall : OK
TransactionControllerTest::testTransferFundsBalanceGood : OK
(gakell) : on quitte le répertoire </users/polabone/Bureau/LOG1000/og1000-06/TP4/RechargeOpusGL_buggy>
```

Dans le bogue qu'on a résolu dans **OpusController::addCredits**.

Si on avait pas vérifié l'état de la variable `dispcredits` n'a jamais été initialisé et cause une erreur qui fait que le notificateur indique qu'il y a 32767 credits d'achetés.

Le “Data Flow Testing” ici a été utile.

E5.C

-Test d'intégration:

Généralement effectué par plusieurs personnes, il consiste en une exécution progressive jusqu'en fin de projet des composantes d'un système en isolant partiellement et en testant les composantes afin qu'elles fonctionnent correctement ensemble au final.

-Tests de Regression

Ce test consiste en répétant des tests existants au fur et à mesure qu'on avance dans le développement du système pour voir si des modifications changent l'état des tests antérieurs qui fonctionnaient auparavant.