**CPP Problem Design**

|  |
| --- |
| **Subject: ATM** |
| **Contributor: 謝宜杭,** **謝公耀, 廖宣瑋** |
| **Main testing concept: Exception Handling**   |  |  | | --- | --- | | **Basics** | **Functions** | | □ C++ BASICS  □ FLOW OF CONTROL  □ FUNCTION BASICS  □ PARAMETERS AND OVERLOADING  □ ARRAYS  □ STRUCTURES AND CLASSES  □ CONSTRUCTORS AND OTHER TOOLS  □ OPERATOR OVERLOADING, FRIENDS,AND REFERENCES  □ STRINGS  □ POINTERS AND DYNAMIC ARRAYS | □ SEPARATE COMPILATION AND NAMESPACES  □ STREAMS AND FILE I/O  □ RECURSION  □ INHERITANCE  □ POLYMORPHISM AND VIRTUAL FUNCTIONS  □ TEMPLATES  □ LINKED DATA STRUCTURES  ■ EXCEPTION HANDLING  □ STANDARD TEMPLATE LIBRARY  □ PATTERNS AND UML | |
| **Description:**   |  | | --- | | A function that returns a special error code is usually better accomplished throwing an exception instead. The following class maintains an account balance.  class Account  {  private:  double balance;  public:  Account()  {  balance = 0;  }  Account(double initialDeposit)  {  balance = initialDeposit;  }  double getBalance()  {  return balance;  }  //returns new balance or -1 if error  double deposit(double amount)  {  if (amount > 0)  balance += amount;  else  return -1; |   return balance;  }  //return new balance or -1 if invalid amount  double withdraw(double amount)  {  if ((amount > balance) || (amount < 0))  return -1;  else  balance -= amount;  return balance;  }  };  Rewrite the class so that it throws appropriate exceptions instead of returning -1 as an error code. Write test code as shown in the following that attempts to withdraw and deposit invalid amounts and catches the exceptions that are thrown.  Note that please use this following code snippets as your main()  //Main  int main()  {  Account a(100);  try  {  cout << "Depositing 50" << endl;  cout << "New balance: " << a.deposit(50) << endl;  //cout << "Depositing -25" << endl;  //cout << "New balance: " << a.deposit(-25) << endl;  cout << "Withdraw 25" << endl;  cout << "New balance: " << a.withdraw(25) << endl;  cout << "Withdraw 250" << endl;  cout << "New balance: " << a.withdraw(250) << endl;  }  catch (InsufficientFunds) // InsufficientFunds: a class name  {  cout << "Not enough money to withdraw that amount." << endl;  }  catch (NegativeDeposit) // NegativeDeposit: a class name  {  cout << "You may only deposit a positive amount." << endl;  }  cout << "Enter a character to exit" << endl;  char wait;  cin >> wait;  return 0;  }  // note that  // class NegativeDeposit {…};  // class InsufficientFunds {…};  **Input:**  **Output:**  **Sample Input / Output：**   |  |  | | --- | --- | | Sample Input | Sample Output | | main.in | sample.out | |
| **■ Eazy,Only basic programming syntax and structure are required.**  **□ Medium,Multiple programming grammars and structures are required.**  **□ Hard,Need to use multiple program structures or more complex data types.** |
| **Expected solving time:**  20 minutes |
| **Other notes:** |