問1　セキュアプログラミングに関する次の記述を読んで，設問1～3に答えよ。

E社は，従業員数100名の自動車部品販売会社であり，販売先は日本各地に散在している。5年前に社内LANを導入した際，営業報告管理システム（以下，Xシステムという）を開発し，社内LAN上で運用を始めた。Xシステムは C++で開発したWebアプリケーション（以下，Xシステムの Web アプリケーションを X-Web アプリという），Webサーバ及びデータベースで構成されている。

〔Xシステムへの新たな要望とその対応〕

これまで営業報告は，その日の営業活動終了後に営業社員が会社に戻り，Xシステムを用いて営業報告書を作成して，更に Xシステム上で営業部長に提出し，必要な場合は口頭によって補足する方法がとられてきた。また，遠方の販売先に出張した場合には電話で報告を行い，後日出社した際に営業報告書を作成して営業部長に提出してきた。

インターネットが普及したことから，会社に戻らなくても出張先で営業報告を行えるようにしてほしいという要望が営業社員から出されていた。さらに，他社との競争も激しくなっていることから，E社としても営業力強化が必要であると経営者が判断し，この要望に対応することになった。具体的には，営業活動の都度，インターネットを利用して報告することで，営業部長が営業活動の状況を逐次把握し，必要に応じて指示ができるようにすることにした。情報システム部のB君が X-Web アプリの改修を担当することになった。

なお，Xシステムのデータベースは現状のものを使うことにした。現状の X-Web アプリは，社外からインターネット経由でアクセスすることを想定しておらず，脆弱(ぜいじゃく)性対策も行われていない。

〔代表的な脆弱性への対策方法の確認〕

社内の他の Web アプリケーションは Java で開発されている。B君は，X-Web アプリを Java で書き直す案もあると考え，先輩の D主任に相談した。D主任から，“一般的には Java が良いと言われているよ”という助言をもらったが，B君は，C++と Java のそれぞれを採用するメリットとデメリットを比較することにした。B君は，X-Web

アプリの中で SQL インジェクションの脆弱性がある関数を一つ見つけたので，C++ と Java での対策方法を比較してみることにした。その関数 getReport の仕様を図 1 に示す。

```
void getReport(sql::Connection *con, string user, string day)
  概要：営業報告を表示する。
  引数：次の三つの引数がある。
        con － データベースにアクセスするためのコネクションオブジェクト
        user － 利用者画面からの入力を基に作成した営業社員名
        day － 利用者画面からの入力を基に作成した，表示したい営業報告の日付情報
  処理：営業社員 user が日付 day の営業活動として登録した全ての営業報告を，データベースへのコネクションオブジェクト con を用いて取得し，その営業報告を表示するための HTML を出力する。
  備考：この関数を呼び出す前処理として，この関数に引き渡す引数の値は，サーバ上の設定情報や利用者画面の入力データから取り出され，セットされている。その他実行に必要な環境も準備されている。
  （省略）
```

図 1　getReport の仕様（抜粋）

SQL インジェクション対策前の C++コードを図 2 に示す。

```
（#include などは省略）
void getReport(sql::Connection *con, string user, string day) {
   string query = "SELECT report FROM reports WHERE user = '" + user + "'";
   query += " AND day = '" + day + "'";
   sql::Statement *stmt;
   sql::ResultSet *res;
   cout << "<HTML><BODY>";
   try {
      stmt = con->createStatement();
      res = stmt->executeQuery(query);
      cout << "user = '" << user << "'<BR>";
      cout << "day = '" << day << "'<BR>";
      while(res->next()) {
         string rep = res->getString("report");
         cout << "report = '" << rep << "'<BR>";
      }
      delete res;
      delete stmt;
   } catch(const Exception& e) {
      cout << "ERROR!";
   }
   cout << "</BODY></HTML>";
}
```

図 2　SQL インジェクション対策前の C++コード

B 君は，このコードに対し，SQL インジェクション対策を行ってみた。この対策は，［ a ］を用いて SQL 文を組み立てるというものである。対策後の C++コードを図 3 に示す。

```
（#include などは省略）
void getReport(sql::Connection *con, string user, string day) {
   string query = "SELECT report FROM reports WHERE user = ? AND day = ?";
   sql::PreparedStatement *pstmt;
   sql::ResultSet *res;
   cout << "<HTML><BODY>";
   try {
      pstmt = con->prepareStatement(query);
      pstmt->setString(1, user);
      pstmt->setString(2, day);
      res = pstmt->executeQuery();
      cout << "user = '" << user << "'<BR>";
      cout << "day = '" << day << "'<BR>";
      while(res->next()) {
         string rep = res->getString("report");
         cout << "report = '" << rep << "'<BR>";
      }
      delete res;
      delete pstmt;
   } catch(const Exception& e) {
      cout << "ERROR!";
   }
   cout << "</BODY></HTML>";
}
```

図 3　SQL インジェクション対策後の C++コード

B 君は，図 3 の C++コードと同様の対策を実施した同じ仕様のプログラムを Java で作成してみた。そのコードを図 4 に示す。

```
（import などは省略）
public void getReport(Connection con, String user, String day) {
   String query = "SELECT report FROM reports WHERE user = ? AND day = ?";
   PreparedStatement ps = null;
   ResultSet rs = null;
   out.write("<HTML><BODY>");
   try {
      ps = con.[ b ](query);
      ps.setString(1, user);
      ps.setString(2, day);
      rs = ps.executeQuery();
      out.write("user = '" + user + "'<BR>");
      out.write("day = '" + day + "'<BR>");
      while (rs.next()) {
         String rep = rs.getString("report");
         out.write("report = '" + rep + "'<BR>");
      }
      ps.close();
   } catch (Exception e) {
      out.write("ERROR!");
   }
   out.write("</BODY></HTML>");
}
```

図4　SQLインジェクション対策を実施したJavaコード

この後，図2～4のコードにはSQLインジェクション以外にも①代表的な脆弱性の原因となるコードが含まれていることが分かり，B君は，[c]を[d]処理するという対策を図3と図4のコードにそれぞれ追加した。

次に，B君は，インターネット上の信頼できるWebサイトを調査し，図5に示すような，脆弱性につながるC++の特性の解説を見つけた。

C++は，メモリ内容への柔軟なアクセスが可能である。ただし，プログラムに次のような問題があっても言語処理系自体では防止できないので，プログラマ自身がこれを回避するコーディングをする必要がある。

脆弱性の要因となる問題

- データを転記する際のメモリ領域の境界チェック漏れ
- [e]による誤った領域へのアクセス
- データへの[e]と，関数への[e]を混同したアクセス
- 整数演算での桁あふれ

上記問題が引き起こす代表的な脆弱性

- [f]
- 整数オーバフロー

図5　脆弱性につながるC++の特性の解説（抜粋）

B 君は，この C++の特性に対して，Java の特性を図 6 に整理した。

Java が提供しているメモリ内容へのアクセス手段は限定的であり，メモリマネジメントは言語処理系の一部である Java VM が行う。長時間連続動作させる場合には [g] の実行による応答性への影響を考慮する必要がある。

図 6　Java の特性

B 君は，以上の調査結果を踏まえ，X-Web アプリの改修でどちらの言語を採用するかを検討するために，両言語を採用する場合のメリット，デメリットを整理した。両言語の比較表を表 1 に示す。

表 1　両言語の比較表（抜粋）

言語	メリット	デメリット
C++	・X-Web アプリで使用している言語なので，改修作業量が少なくなる。 ・処理が高速である。	・利用可能な既存のアプリケーションフレームワークが少ない。 ・特に [f] に関係した脆弱性対策が必要である。
Java	・利用可能な既存のアプリケーションフレームワークが豊富である。 ・多くの環境で動作可能である。	・X-Web アプリで使用していない言語なので，改修作業量が多くなる。 ・[g] への対応が必要な場合がある。

〔その後の状況〕

今回の対応では，C++を用いた場合に [f] 対策が多くの部分で必要となることが確認されたので，表 1 を基に B 君は Java を採用することを提案した。その提案を受けて，情報システム部内で検討を重ね，情報システム部長が Java を採用することを承認した。言語を変更したことで X-Web アプリを書き直すことになったが，B 君はこの改修を無事に行うことができた。

設問1　図2～4のコードについて，(1)～(6)に答えよ。

(1) 図2でSQLインジェクションの原因となるコーディング部分が複数ある。そのコーディング部分に書かれた変数名を，コード上から選び，全て答えよ。

(2) 本文中の［　a　］に入れる適切なプログラミング技法を12字以内で答えよ。

(3) 図4中の［　b　］に入れる適切な文字列を英字で答えよ。

(4) 本文中の下線①の代表的な脆弱性を解答群の中から選び，記号で答えよ。

解答群

ア　クロスサイトスクリプティング

イ　クロスサイトリクエストフォージェリ

ウ　コマンドインジェクション

エ　セッションフィクセーション

(5) 本文中の［　c　］，［　d　］について，［　d　］に入れる適切なプログラミング技法を10字以内で，その技法の対象を明確に示す［　c　］に入れる適切な字句を20字以内で答えよ。

(6) 上記(5)の技法の対象となる変数が図3中に複数ある。その変数名を図3中から選び，全て答えよ。

設問2　図5中の［　e　］に入れる適切な字句を6字以内で，図5中，表1中及び本文中の［　f　］，並びに図6中及び表1中の［　g　］に入れる適切な字句を，それぞれ12字以内で答えよ。

設問3　図6及び表1には，脆弱性を狙った［　f　］攻撃がC++では問題となるのに，Javaでは問題とならない根本的な理由が具体的に記述されていない。図5の内容を踏まえて，根本的な理由であるJavaの言語仕様上の特徴を20字以内で述べよ。