1. In pascal verses C both strings are viewed as basic arrays of characters for both. The formatting of the array of strings is the main difference between the two of them. In pascal the strings have the 0 index of the char array as the length for the array and define how long the string is. For C the string start in the 0 index spot but is always terminated by a Null-Character.  
   For the C vs C++ that major difference between the 2 are that on is a basic array of characters and the other is its own class or object. This detail is important because in C the concept and functionality “strings” can’t be modified as they are basic character arrays, while in C++ the string class can be modified for special situations.  
   For Java vs C++ the strings are pretty similar in that they are both their own object. 1 difference is that the characters in a java string is made up of 16-bit Unicode while for C++ the actually character stored is platform dependent. Another Large difference between the two is how they work with operands such as + and ==. In Java adding a string to and old string and setting it equal to that old string actually takes both strings and adds them into a new empty string and then changes the old string pointer to the newly created string. Also in java the expression: “bacon” == “bacon” is false. The reason for this is that == operand is looking at the two strings as two different objects. This is way the .equals() method is often used in java. In C++ this is the opposite.
2. It seems that the general issue with security risks in C is writing the string outside of a defined container length. Since the end of a string is supposed to be Null terminated if a string is to large for its container the null termination will be cut allowing possible for iterating pass the strings decided length. With Pascal the sting length is place in the first character slot. So the length of a string is ways know and maxed out at a fixed point. With c there also is an issue it in general as it is very pointer based which can cause a lot of issues with the wrong data getting pointed too and passed around.
3. UTF-8 Seems to be the most ideal, it is backward compatible with asci and can support several thousands of characters since it is 1-4 bytes long. Thus this should be able to handle new age media and old server based solutions.