

**课程小作业报告**

**计算思维：数据结构与算法**

|  |  |
| --- | --- |
| **课程名称：** | **计算机与软件工程概论** |
| **学生姓名：** | **快要变成猪了QwQ** |
| **学生学号：** | **XXX** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2023-2024学年第1学期** |

**软件学院**

**2023年10月**

**课程作业2：计算思维**

## 问题：生成密码空间

1. 给你一个字符集合：{a, b, c} // 集合中的元素是不能重复的
2. 定义，字符串密码为：
   1. 不能为空，即必须至少包含字符集合中的一个字符；
   2. 仅由字符集合中的元素串接而成，比如，字符串密码ab，由字符a与b，按a在前，b在后的顺序串接而成；
   3. 字符串密码的长度len，定义为它所包含的字符的个数；
   4. 在长度为len（len>=2）的字符串密码中，字符是可以重复使用的，比如，aa，或者aab。

**使用递归思维，设计一个解决方案passgen：**

1. 输入：字符集合{ a, b, c }，和字符串密码的最大长度len，比如len=5；
2. 输出：所有可能的字符串密码集合。

## 1、知识与技术总结

#### 1.1 什么是数据结构？

A data structure is a way of **organizing and storing data in a computer so that it can be efficiently accessed and manipulated. It defines the layout and organization of data in a specific format**, which enables efficient operations such as insertion, deletion, searching, and sorting.

There are various types of data structures, each with its own characteristics and use cases. Here are a few examples:

1. **Arrays**: An array is a collection of elements of the same data type arranged in a contiguous block of memory. It allows efficient random access to elements using their indices. For example, an array can store a list of integers or strings.

2. **Linked Lists**: A linked list is a linear data structure composed of nodes, where each node contains a data element and a reference (or link) to the next node. It provides efficient insertion and deletion at any position in the list. Linked lists are commonly used in scenarios where frequent insertions and deletions are required.

3. **Stacks**: A stack is a last-in, first-out (LIFO) data structure. It follows the principle of "last in, first out," similar to a stack of plates. Elements can be added or removed only from the top of the stack. Stacks are used in various scenarios, such as function call stacks, expression evaluation, and undo mechanisms.

4. **Queues**: A queue is a first-in, first-out (FIFO) data structure. It follows the principle of "first in, first out," like a queue of people waiting in line. Elements can be added at the rear (enqueue) and removed from the front (dequeue) of the queue. Queues are commonly used in scenarios such as task scheduling, breadth-first search, and buffering.

5. **Trees**: A tree is a hierarchical data structure consisting of nodes connected by edges. It has a root node at the top and child nodes branching out from the parent nodes. Trees are used in many applications, such as representing hierarchical relationships, organizing data hierarchically, and implementing search algorithms like binary search trees and AVL trees.

6. **Graphs**: A graph is a non-linear data structure composed of nodes (vertices) connected by edges. It represents relationships between objects and is widely used in applications involving networks, social media analysis, route planning, and more. Graphs can be directed (edges have a specific direction) or undirected.

These are just a few examples of data structures, and there are many more, each with its own advantages and use cases. The choice of a data structure depends on the specific requirements of the problem and the efficiency needed for various operations on the data.

#### 1.2 什么是算法？

An algorithm is **a step-by-step procedure or set of rules for solving a specific problem or accomplishing a particular task.** **It is a well-defined sequence of instructions designed to perform a specific computation or operation.** Algorithms serve as the core building blocks of computer programs and are essential for solving complex problems efficiently.

Here are a few examples of algorithms:

1. **Binary Search**: This algorithm is used to search for a target element in a sorted list of elements. It repeatedly divides the list in half and compares the target element with the middle element, narrowing down the search range until the target element is found or determined to be absent.

2. **Dijkstra's Algorithm**: This algorithm is used to find the shortest path between a source node and all other nodes in a weighted graph. It maintains a set of tentative distances from the source node to all other nodes and iteratively selects the node with the shortest tentative distance, updating the distances of its neighboring nodes. It continues this process until all nodes have been visited.

3. **QuickSort**: This algorithm is a divide-and-conquer sorting algorithm. It selects a pivot element from the list and partitions the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. It recursively applies the same process to the sub-arrays until the entire list is sorted.

4. **Depth-First Search (DFS)**: This algorithm is used to traverse or search a graph or tree data structure. It explores as far as possible along each branch before backtracking. DFS is often used to solve problems such as finding connected components, detecting cycles, and solving puzzles like the maze problem.

5. **Breadth-First Search (BFS)**: This algorithm is used to traverse or search a graph or tree data structure. It explores all the neighboring nodes of the current node before moving to the next level. BFS is often used to solve problems such as finding the shortest path, analyzing social networks, and solving puzzles like the sliding tile puzzle.

6. **Dynamic Programming (DP)**: Dynamic programming is a technique used to solve complex problems by breaking them down into overlapping subproblems and solving them in a bottom-up manner. It stores the solutions to subproblems in a table and reuses them to solve larger subproblems or the main problem. DP is often used to solve optimization problems, such as finding the longest common subsequence, the maximum sum subarray, or the optimal path in a grid.

These examples illustrate a range of algorithms that are commonly used in various domains, showcasing their diverse applications in searching, sorting, path finding, and optimization.

These are just a few examples of algorithms, and there are countless others that address various computational problems. Algorithms can be simple or complex, and their efficiency and correctness play a crucial role in solving problems effectively.

#### 1.3 什么是递归？

Recursion is a programming technique where **a function calls itself to solve a smaller instance of the same problem.** In other words, **a recursive function solves a problem by reducing it into smaller, simpler versions of the same problem.**

Let's take the example of **calculating the factorial** of a non-negative integer using recursion.

The factorial of a non-negative integer n, denoted as n!, is the product of all positive integers from 1 to n. The factorial can be calculated recursively using the following formula: n! = n \* (n-1)!, with the base case being 0! = 1.

For example, if we call factorial(5), the function will call itself with the values factorial(4), factorial(3), factorial(2), factorial(1), and finally factorial(0). When n becomes 0, the base case is triggered, and the function starts returning values: factorial(0) returns 1, factorial(1) returns 1 \* 1 = 1, factorial(2) returns 2 \* 1 = 2, factorial(3) returns 3 \* 2 = 6, and factorial(4) returns 4 \* 6 = 24. Finally, factorial(5) returns 5 \* 24 = 120, which is the factorial of 5.

This recursive approach breaks down the problem of calculating the factorial into smaller subproblems (calculating the factorial for smaller values of n) until reaching the base case. Recursion can be a powerful technique for solving problems that exhibit a recursive structure.

## 2、问题求解

#### 2.1 问题分析

Suppose we have variables n, len and an array c[100] to store the length of our character set, the maximum length of the generated password and elements in the character set, respectively. One of the most plain approach is using nested loops at a level of “len”. Granted, it works but it does not work that gracefully. To improve readability and to minimize our code, we can define a recursive function that calls itself to achieve the effect of nested loops at multiple levels.

The overall task can be divided into two smaller subtasks, which allows for our recursive method. One of the subtasks is to generate a single bit of password for the MSB(the Most Significant Bit) and another subtask is to generate the rest of the password with a length of len-1“. Therefore, the overall task can be divided into several fundamental operations, i.e., output every possible character for every individual bit.

#### 2.2 伪代码

Void passgen()

{

//check if the boundary condition is met.

//Subtask1 generate the MSB

//subtask2 handle the rest of the password, i.e., call passgen(len-1).

}

Int main()

{

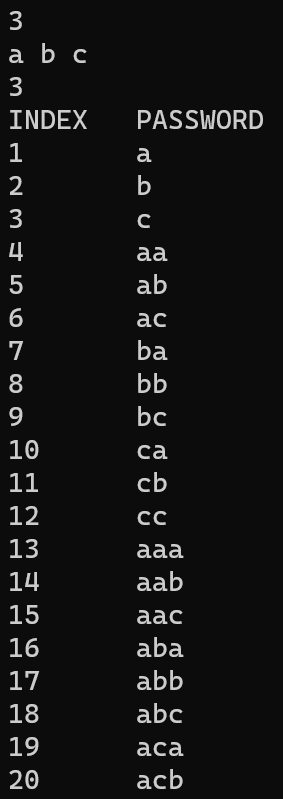
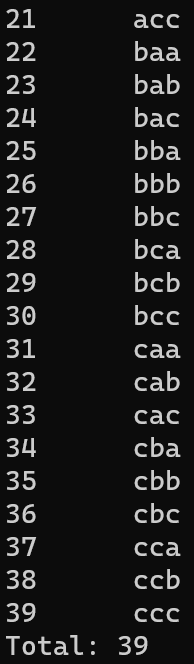
//input n, c[], len.

//call function passgen(len).

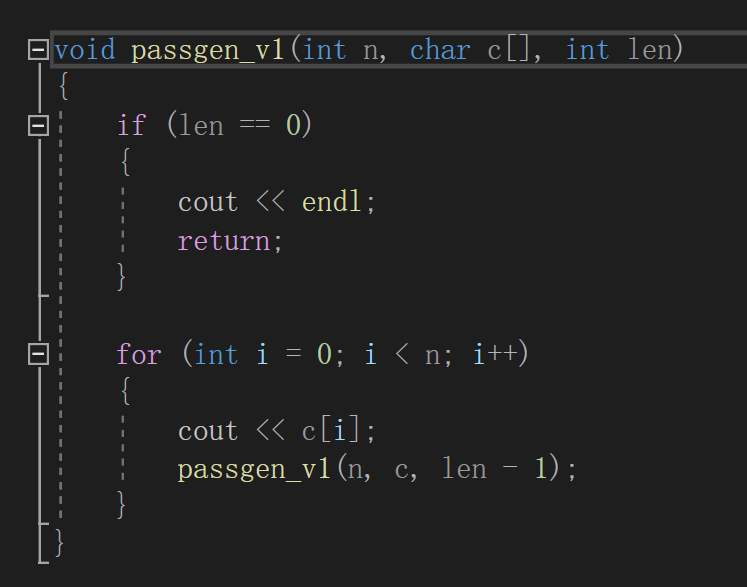
Return 0;

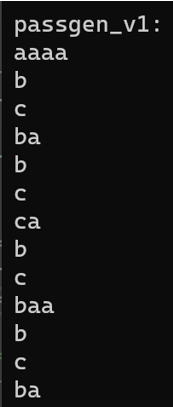
}

#### 2.3 算法输出及结果分析

## 3、总结



Above was my first version of Passgen function, I directly output every specific bit which resulted to an awkward outcome:

Let’s look closer at our weird outcome. Notice that our outcome is the result of “Backtracking Algorithm” because if we add 3 ‘a’s in front of the second and third line, then we obtain the right answer. To tackle this problem, I add an array to store previously obtained bits and output them all at once if the remaining length is 0.

## 参考

IDE: VS2022 C++ Language Standard: C++14 OS: Windows11

“Solution.cpp” is source code of the project.