Wykonał: Radosław Smoter

Grupa: 14

Nr: 27

Numer zadania: 8

Przykład: 62

Prowadzący: Prof. dr hab. inż. Volodymyr Samotyy

Politechnika Krakowska

Wydział Inżynierii Elektrycznej i Komputerowej

Sprawozdanie: Wstęp do Programowania

Kraków 2021

Spis treści

Polecenie

Wygenerować permutacje według niżej podanych specyfikacji.

Numer zadania	62
Zbiór	literalny (a, b, c, …)
Algorytm permutacji	1.11
Ilość elementów zbioru	6

Kod programu

```c
/**
 * @file main.c
 * @author Radoslaw Smoter (radoslaw.smoter@student.pk.edu.pl)
 * @version 0.1
 * @brief Recursive Heap's Algorithm
 * @date 2021-12-17
 *
 * @copyright Copyright (c) 2021
 */

/*
  * Generate permutations considering letters with 6 elements in a set
  * Permutation algorithm: 1.11
*/


#include <stdio.h>
#include <stdlib.h>


/* Number of elements of the permutation */
#define NUMBER_OF_LETTERS  6
/* Single permutation */
char P[NUMBER_OF_LETTERS];

/* Colors */
#define C_BRED "\e[1;31m"
#define C_RESET "\e[0m"
#define C_BYEL "\e[1;33m"


/* Exchange values of two elements of an array */
void exch(char v[], int i, int j) {
  char temp = v[i];
  v[i] = v[j];
  v[j] = temp;
}


/* Return true if number if even, false otherwise */
_Bool isEven(int n) { return !(n % 2); }


/* Generate permutations. m is the number of initial elements of the permutation set. m
= NUMBER_OF_LETTERS initially. Later, m < NUMBER_OF_LETTERS. Performs permutations by
calling itself with m-th element unaltered and then m-1 times with m-th element
exchanged for one of initial m-1 elements. */
void PERM(int m) {
  /* Number of permutation */
  static int noPerm = 1;
  /* Value of m and i on previous iteration. For colors. */
  static int mval = -1;
  static int ival = -1;
```

```
  /* Base case - print results */
  if (m == 1) {
    printf("%5i:\t", noPerm++);
    for (int i = 0; i < NUMBER_OF_LETTERS; i++) {
      /* Highlight transpositions. */
      if (i == mval)
        printf("%s%5c%s", C_BRED, P[i], C_RESET);
      else if (i == ival)
        printf("%s%5c%s", C_BYEL, P[i], C_RESET);
      else
        printf("%5c", P[i]);
    }
    printf("\n");
  } else {
    PERM(m - 1);

    /* m-th element swapped with each m-1 */
    for (int i = 0; i < m-1; i++) {
      /* Swap dependent on parity of m (even) */
      if (isEven(m))
        exch(P, i, m - 1);
      /* Odd */
      else
        exch(P, 0, m-1);
      mval = m - 1;
      ival = i;
      PERM(m - 1);
    }
  }
}


/* Calculate factorial of given number */
long long factorial(int n) {
  if (n < 0) return 0;
  else if (n == 0 || n == 1) return 1;
  return n * factorial(n - 1);
}


int main(void)
{
  for (int i = 0; i < NUMBER_OF_LETTERS; i++) P[i] = i + 'a';

  PERM(NUMBER_OF_LETTERS);
  printf("Predicted number of permutations: %lli\n", factorial(NUMBER_OF_LETTERS));

  return 0;
}
```

Wyniki

Pomijam dla zwięzłości (720 wyników).

Opis programu

Funkcja main() tego programu składa się z pętli for(), która zapełnia tablicę P[] znakami alfabetu. Następnie wykonuje się funkcja PERM(), która przeprowadza permutacje. Na koniec, wypisywana jest przewidywana ilość permutacji, jaka powinna się wykonać (dla 6 to 6!=720).

Funkcja PERM() przyjmuje w parametrze ilość permutacji, jaką ma wykonać. W zmiennej noPerm przechowywany jest aktualny numer permutacji, zwiększany co każdą permutację. Jeżeli parametr m wyniesie 1, to wypisywany jest wynik permutacji. W przeciwnym przypadku następuje rekurencja – algorytm odwołuje się do siebie samego z m-tym elementem niezmienionym, a następnie m-1 razy z m-tym elementem zmienionym. Zmiana dokonuje się na podstawie parzystości liczby m. Dla parzystych zamienia się elementy o indeksach i z m-1, dla nieparzystych o indeksach 0 z m-1.

Wnioski

Prawdziwość podanego algorytmu wynika z treści zadania. Uzyskane wyniki dają satysfakcjonującą ilość permutacji, na czego podstawie można przypuszczać o poprawności implementacji.