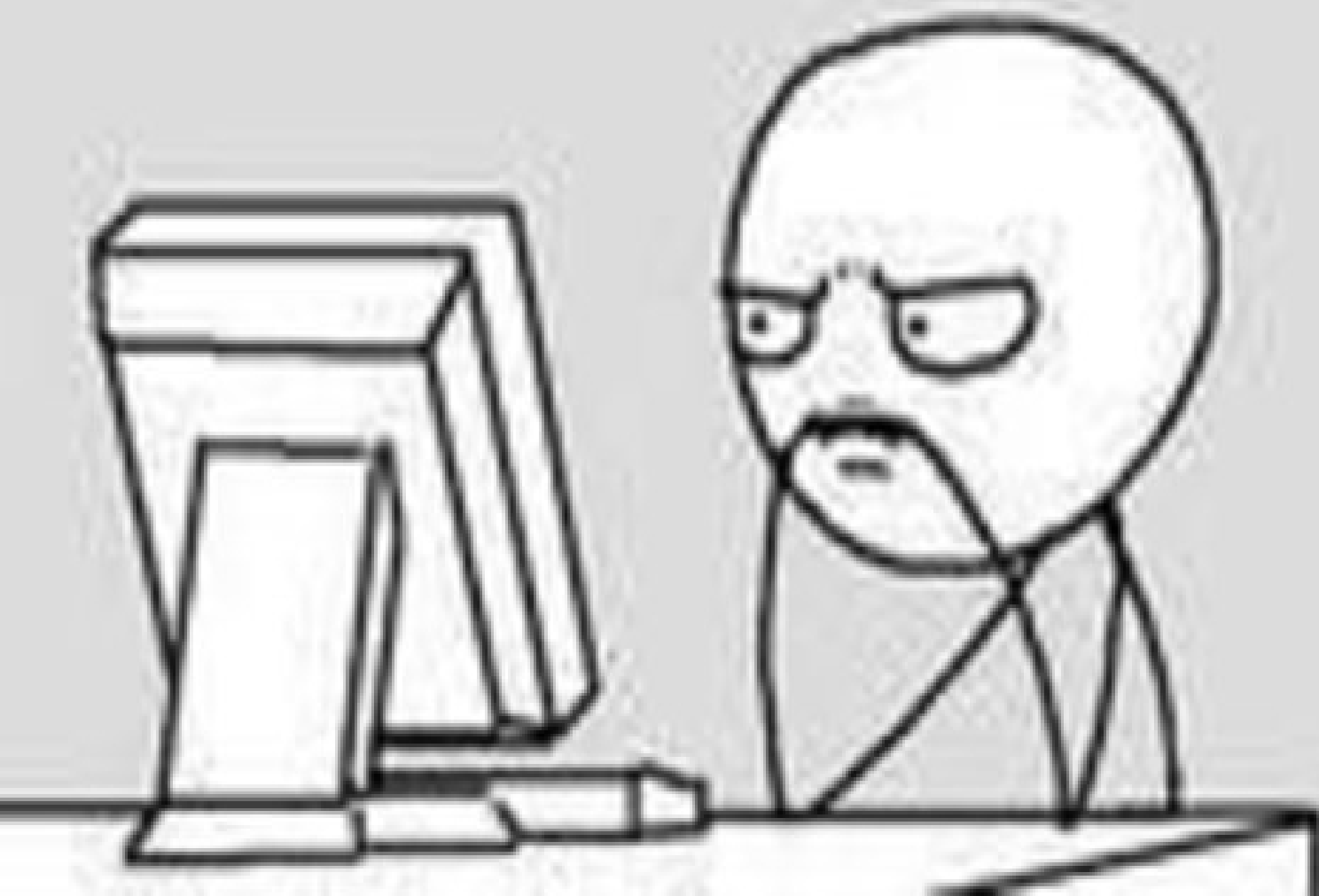


# Combinatória



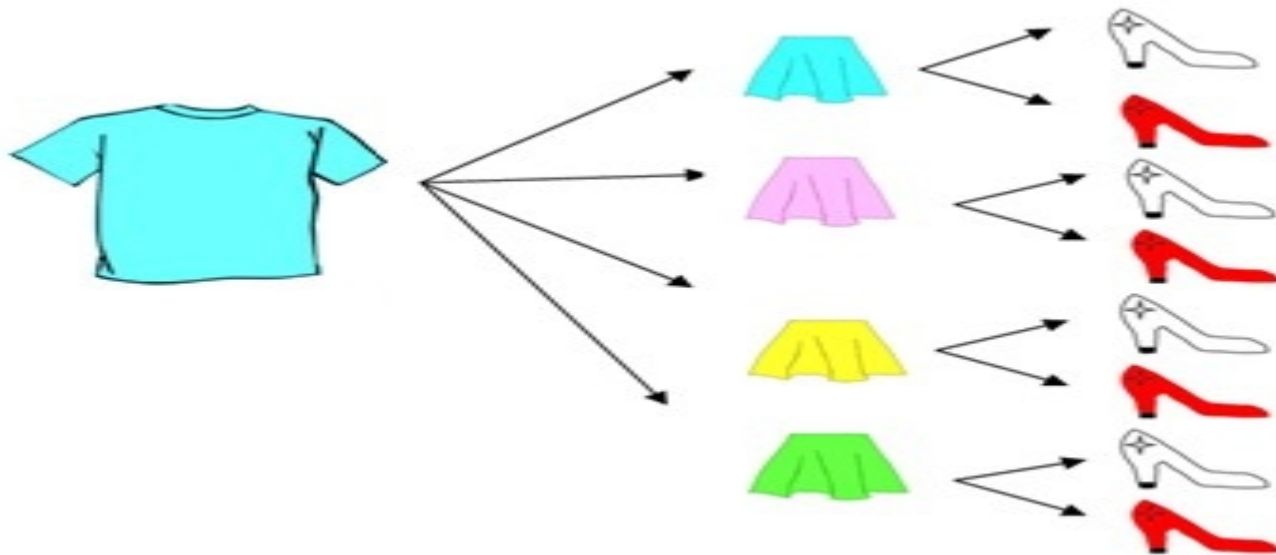
# DEFINIÇÃO :

- A combinatória é um ramo da matemática que estuda coleções finitas de elementos que satisfazem critérios específicos determinados, e se preocupa, em particular, com a contagem de elementos nessas condições.



# PRINCÍPIO FUNDAMENTAL DA CONTAGEM

- Equivale ao considerar que determinada atividade pode ser realizada em duas etapas, ou seja, de  $M$  e  $N$  maneiras. Dessa forma, o resultado é dado pelo produto de  $M$  por  $N$ .



# PERMUTAÇÃO SIMPLES

- Permutar significa trocar, ou seja, um arranjo simples de um conjunto  $n$ .

Exemplo:  $A = \{a_1, a_2, a_3, \dots, a_n\}$

$$P_n = A_{n,n} = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n!$$

Tal ponto de exclamação (!) representa o fatorial desse número, sendo ele obtido pelo produto de seus antecessores positivos e maiores que zero.

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \dots 3 \cdot 2 \cdot 1$$



# PERMUTAÇÃO COM REPETIÇÃO

- De um modo geral, acontece quando, dados  $n$  elementos, dois ou mais são iguais entre si.
- Considere a palavra MATEMÁTICA

$$P_{10}^{3,2,2} = \frac{10!}{2! \cdot 3! \cdot 2!} = \frac{3628800}{2 \cdot 1 \cdot 3 \cdot 2 \cdot 1 \cdot 2 \cdot 1} = \frac{3628800}{24} = 151200$$

Dessa forma, a equação de permutação com repetição é denotada como :

$$P_n^{\alpha, \beta, \gamma} = \frac{n!}{\alpha! \beta! \gamma!}$$



# ARRANJO SIMPLES

- Dado um conjunto  $A = \{a_1, a_2, a_3, \dots, a_n\}$  com  $n$  elementos distintos, chamaremos de arranjo simples toda a sequência formada por uma quantidade delimitada de elementos do conjunto onde a ordem dos elementos faz diferença.
- Sua fórmula é dada por :

$$A_{n,p} = \frac{n!}{(n-p)!}$$



# COMBINAÇÃO SIMPLES

- Pode ser definida como um agrupamento de elementos de um subconjunto, onde a ordem dos elementos não é considerada na formação de subconjuntos.
- Sua fórmula é dada por :

$$C_{n,p} = \frac{n!}{p! \cdot (n-p)!}$$



# PROBABILIDADE

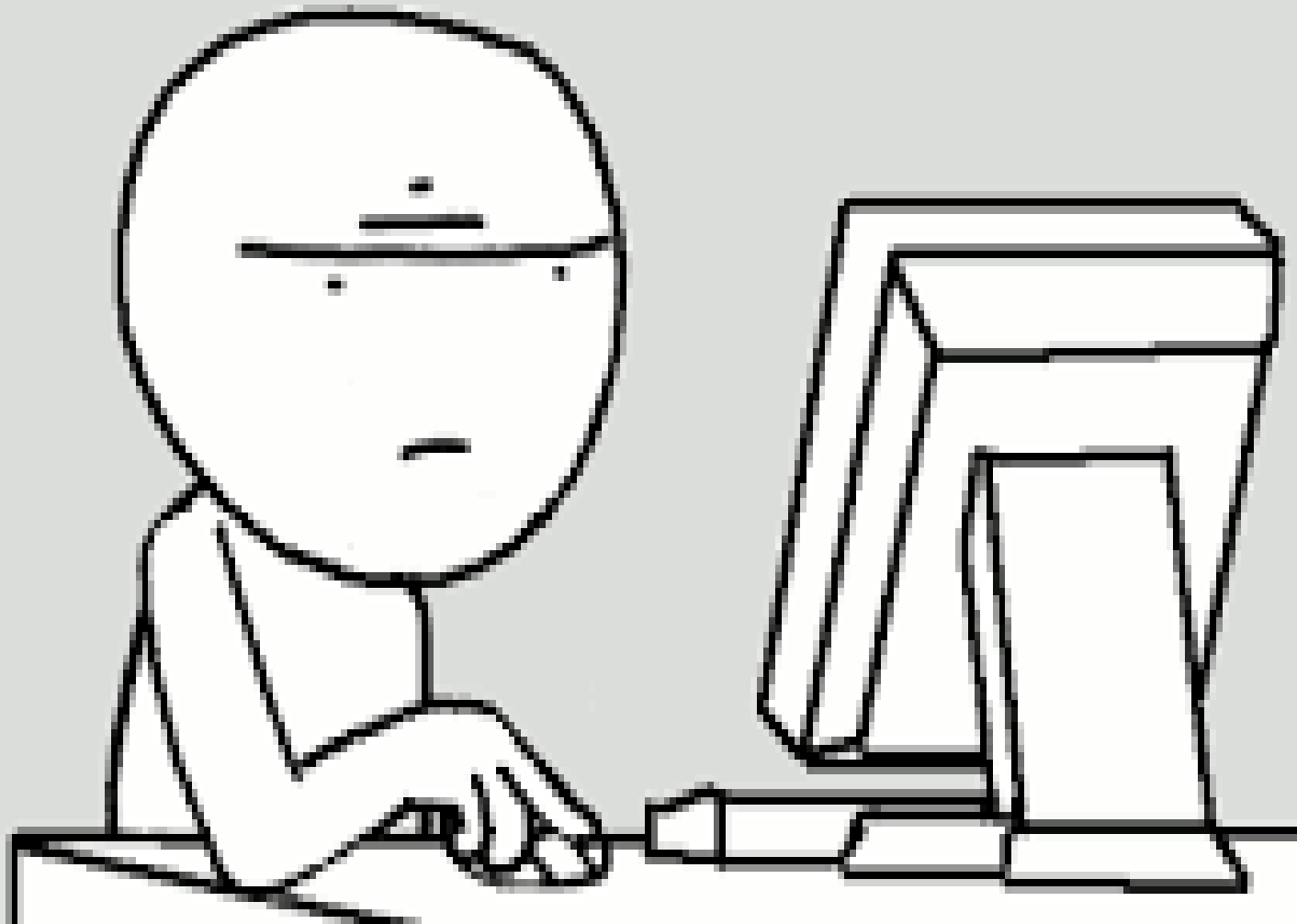
- Pode ser compreendido como o estudo das chances de obtenção de cada resultado de um experimento aleatório. A essas chances são atribuídos os números pertencentes ao intervalo de 0 e 1, tendo maior chance de ocorrer aqueles que tendem a 1.

$$F(total) = F(0) + F(> 0)$$





# Combinações em C++



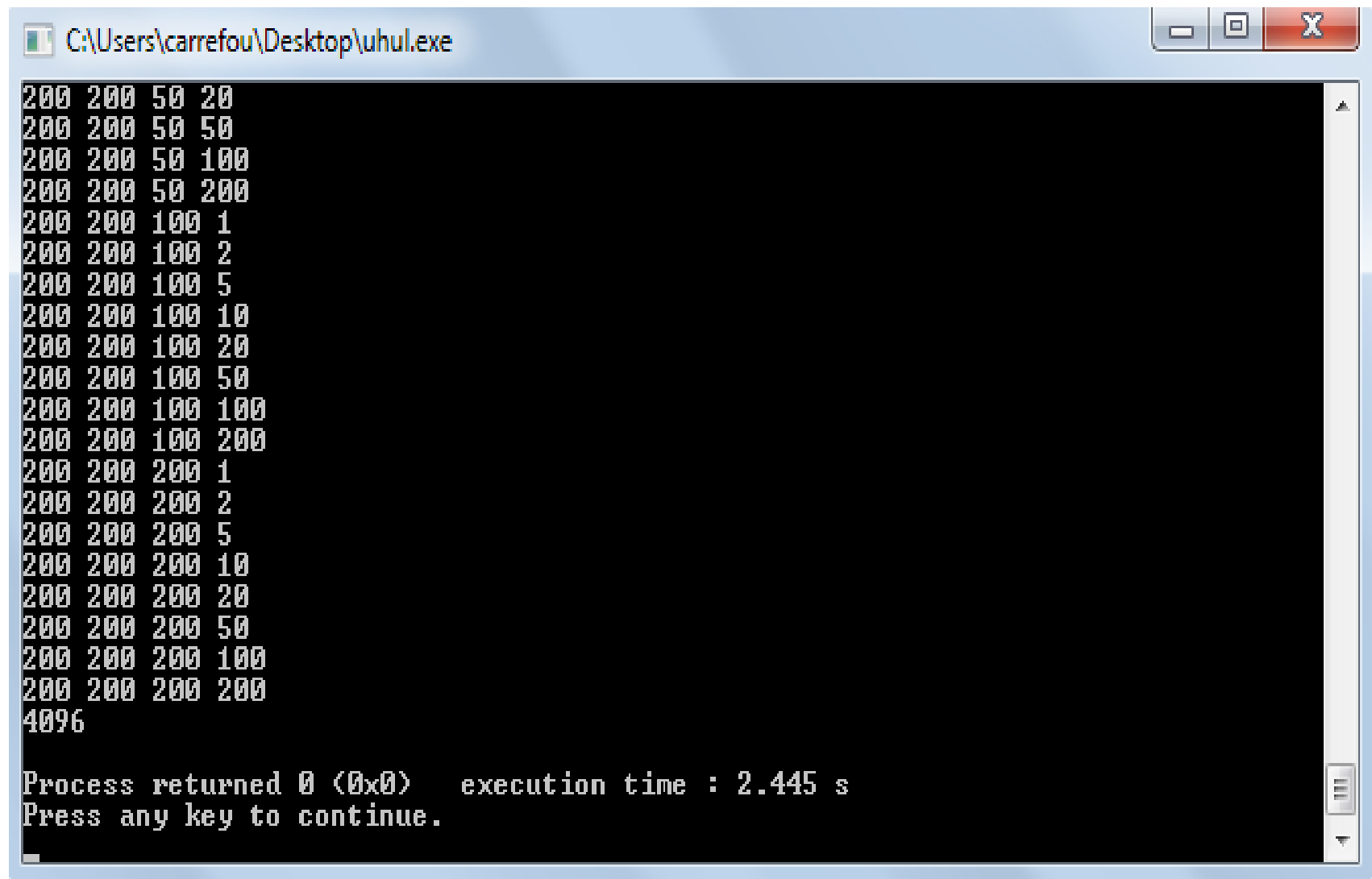
- Imaginemos agora que pretendemos, dispondo de quantidades infinitas de cada tipo de moeda, calcular as possibilidades de determinar o valor de qualquer conjunto de quatro moedas possível, fazendo duas análises, uma com repetição de moedas e outra sem tal repetição .



- 1º - Com repetição – Podemos repetir moedas iguais

```
Start here X *combinatoria.c X aa.cpp X uhul.cpp X
1  #include <iostream>
2  using namespace std;
3  int main(){
4      int n = 8;
5      int moedas[8] = {1, 2, 5, 10, 20, 50, 100, 200};
6      int l0, l1, l2, l3;
7      int total = 0;
8
9      for (l0=0; l0<n; l0++){
10         for (l1=0; l1<n; l1++){
11             for (l2=0; l2<n; l2++){
12                 for (l3=0; l3<n; l3++){
13                     cout << moedas[l0] << " " << moedas[l1] << " " << moedas[l2] << " " << moedas[l3] << endl;
14                     total++;
15                 }
16             }
17         }
18     }
19     cout << total << endl;
20     return 0;
21 }
22
```

- Saída :



```
C:\Users\carrefou\Desktop\uhul.exe

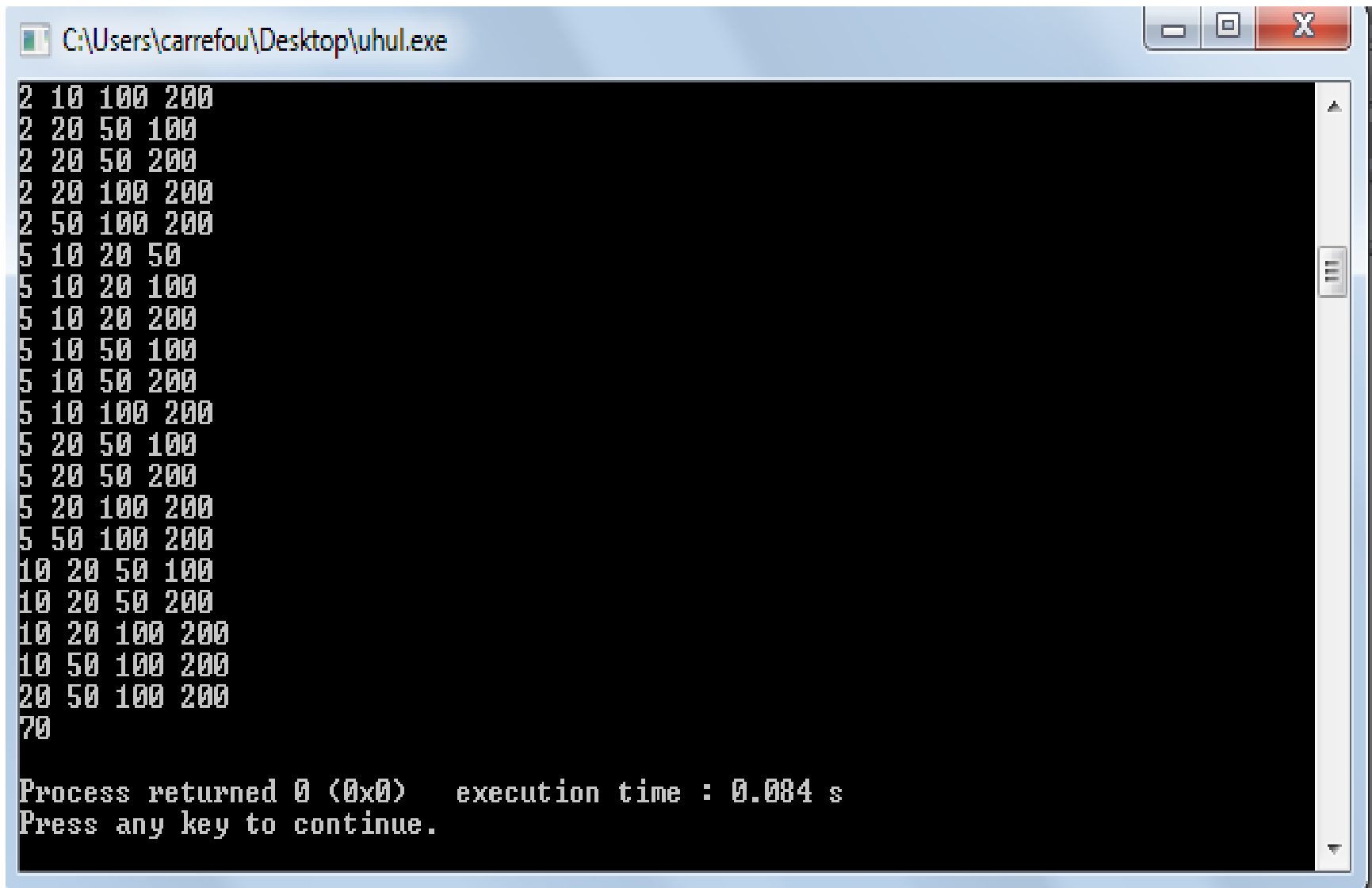
200 200 50 20
200 200 50 50
200 200 50 100
200 200 50 200
200 200 100 1
200 200 100 2
200 200 100 5
200 200 100 10
200 200 100 20
200 200 100 50
200 200 100 100
200 200 100 200
200 200 200 1
200 200 200 2
200 200 200 5
200 200 200 10
200 200 200 20
200 200 200 50
200 200 200 100
200 200 200 200
4096

Process returned 0 (0x0)   execution time : 2.445 s
Press any key to continue.
```

- 2º - Sem repetição – Não podemos repetir as moedas

```
Starthere X *combinatoria.c X aa.cpp X *uhul.cpp X
1  #include <iostream>
2  using namespace std;
3  int main(){
4      int n = 8;
5      int moedas[8] = {1, 2, 5, 10, 20, 50, 100, 200};
6      int i, j, k, l;
7      int total = 0;
8
9      for (i=0; i<n-3; i++){
10         for (j=i+1; j<n-2; j++){
11             for (k=j+1; k<n-1; k++){
12                 for (l=k+1; l<n; l++){
13                     cout << moedas[i] << " " << moedas[j] << " " << moedas[k] << " " << moedas[l] << endl;
14                     total++;
15                 }
16             }
17         }
18     }
19     cout << total << endl;
20     return 0;
21 }
22
```

## ○ Saída :



```
C:\Users\carrefou\Desktop\uhul.exe
2 10 100 200
2 20 50 100
2 20 50 200
2 20 100 200
2 50 100 200
5 10 20 50
5 10 20 100
5 10 20 200
5 10 50 100
5 10 50 200
5 10 100 200
5 20 50 100
5 20 50 200
5 20 100 200
5 50 100 200
10 20 50 100
10 20 50 200
10 20 100 200
10 50 100 200
20 50 100 200
70

Process returned 0 (0x0)   execution time : 0.084 s
Press any key to continue.
```

## < ALGORITHM >

- Algumas funções de permutação já definidas em C++ se encontram nessa biblioteca.
- Sua declaração é feita como qualquer outra, veja:  
`#include <algorithm>`
- Entretanto, essa mesma biblioteca está inserida em outra biblioteca, a `<bits/stdc++.h>` , ou seja, ela engloba todas as suas funções e pode ser usada ao invés de `<algorithm>`.



# NEXT\_PERMUTATION

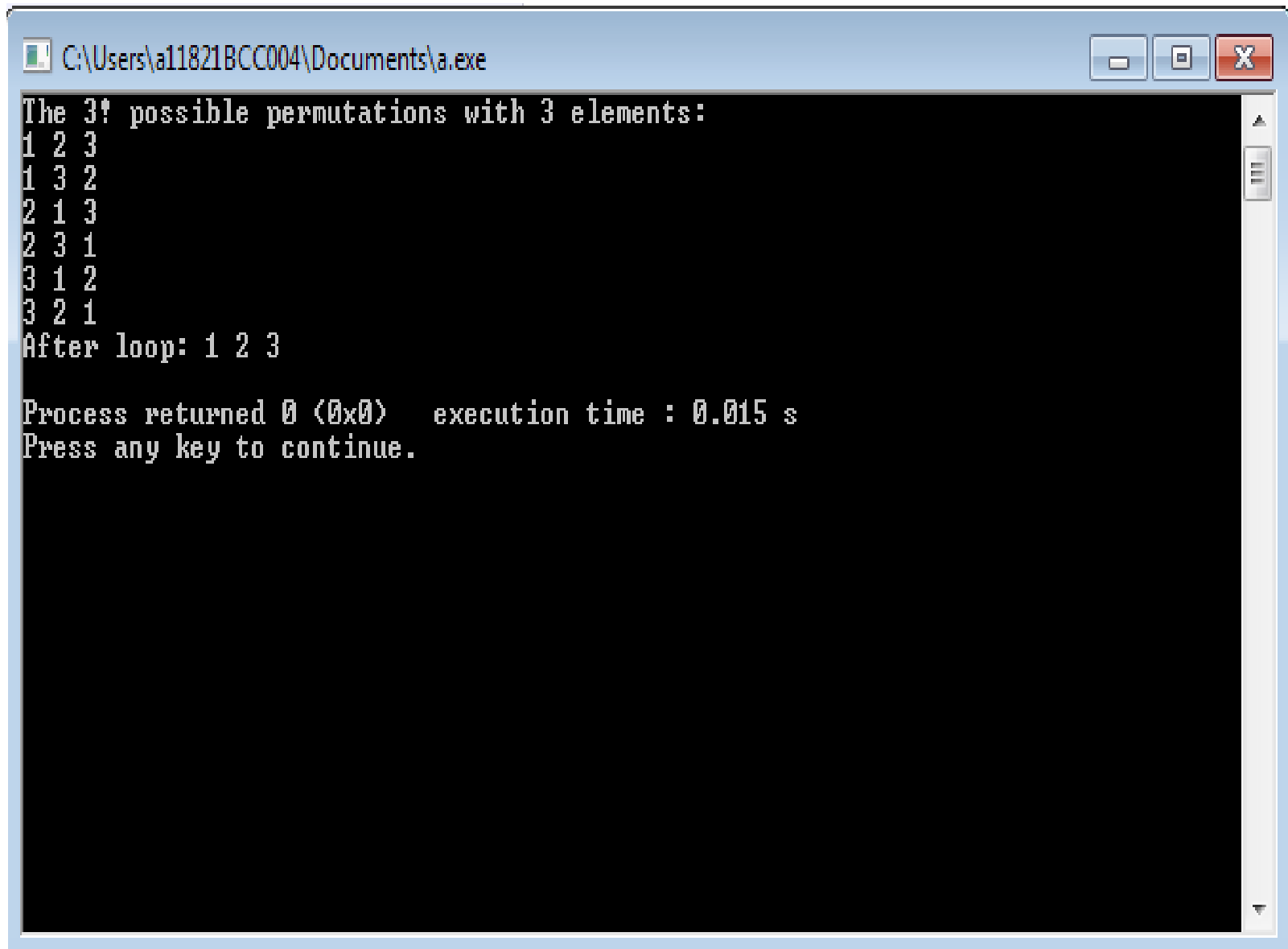
- Reorganiza os elementos no intervalo [first,last) na próxima permutação lexicograficamente maior, onde uma permutação é cada um dos possíveis  $N$  arranjos dos elementos do intervalo.

```
Start here  X  a.cpp  X
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main () {
5      int myints[] = {1,2,3};
6
7      sort (myints,myints+3);
8
9      cout << "The 3! possible permutations with 3 elements:\n";
10     do {
11         cout << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';
12     } while ( next_permutation(myints,myints+3) );
13
14     cout << "After loop: " << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';
15
16     return 0;
17 }
18
```





- Saída :



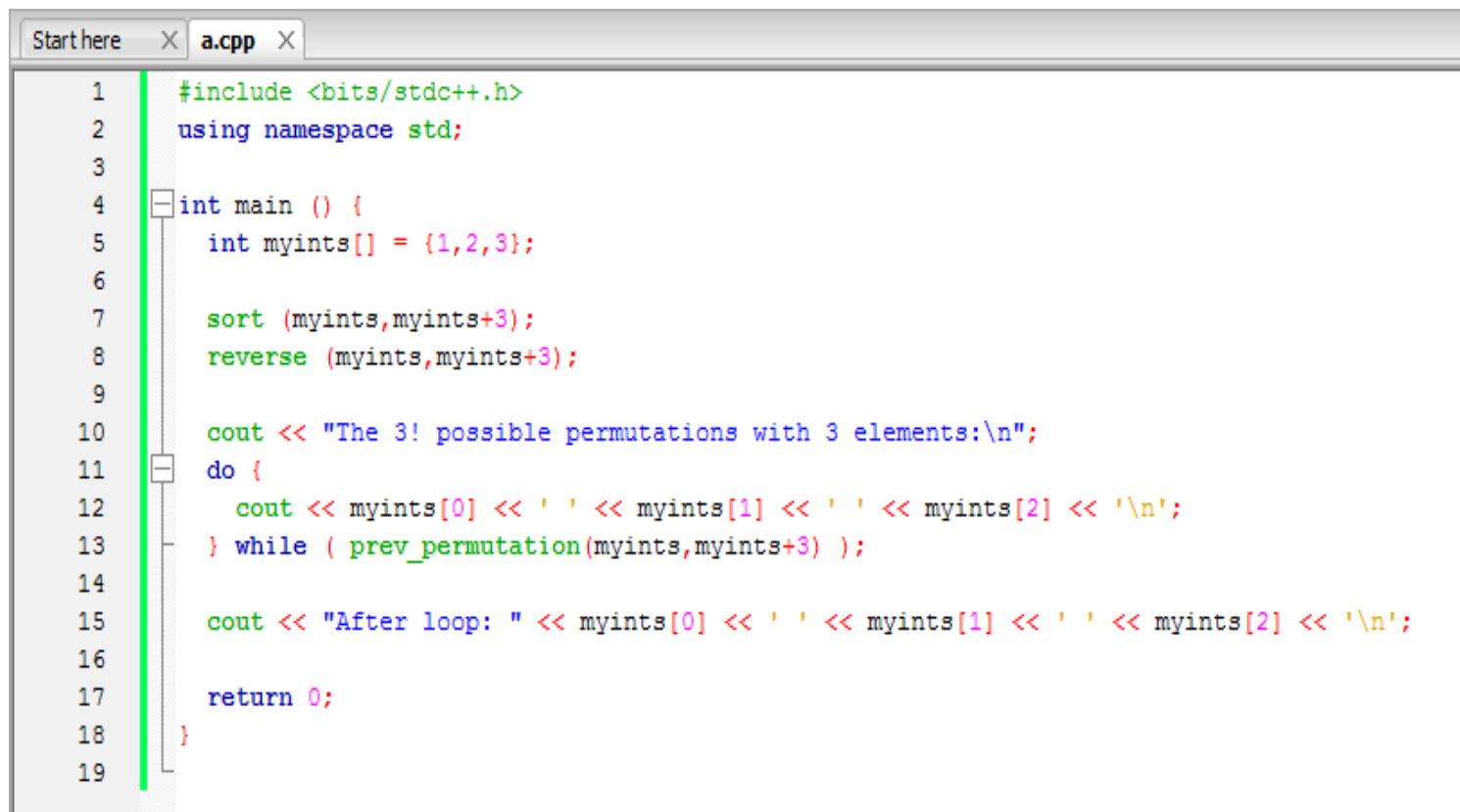
```
C:\Users\al1821BCC004\Documents\>a.exe

The 3! possible permutations with 3 elements:
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
After loop: 1 2 3

Process returned 0 (0x0)   execution time : 0.015 s
Press any key to continue.
```

## PREV\_PERMUTATION

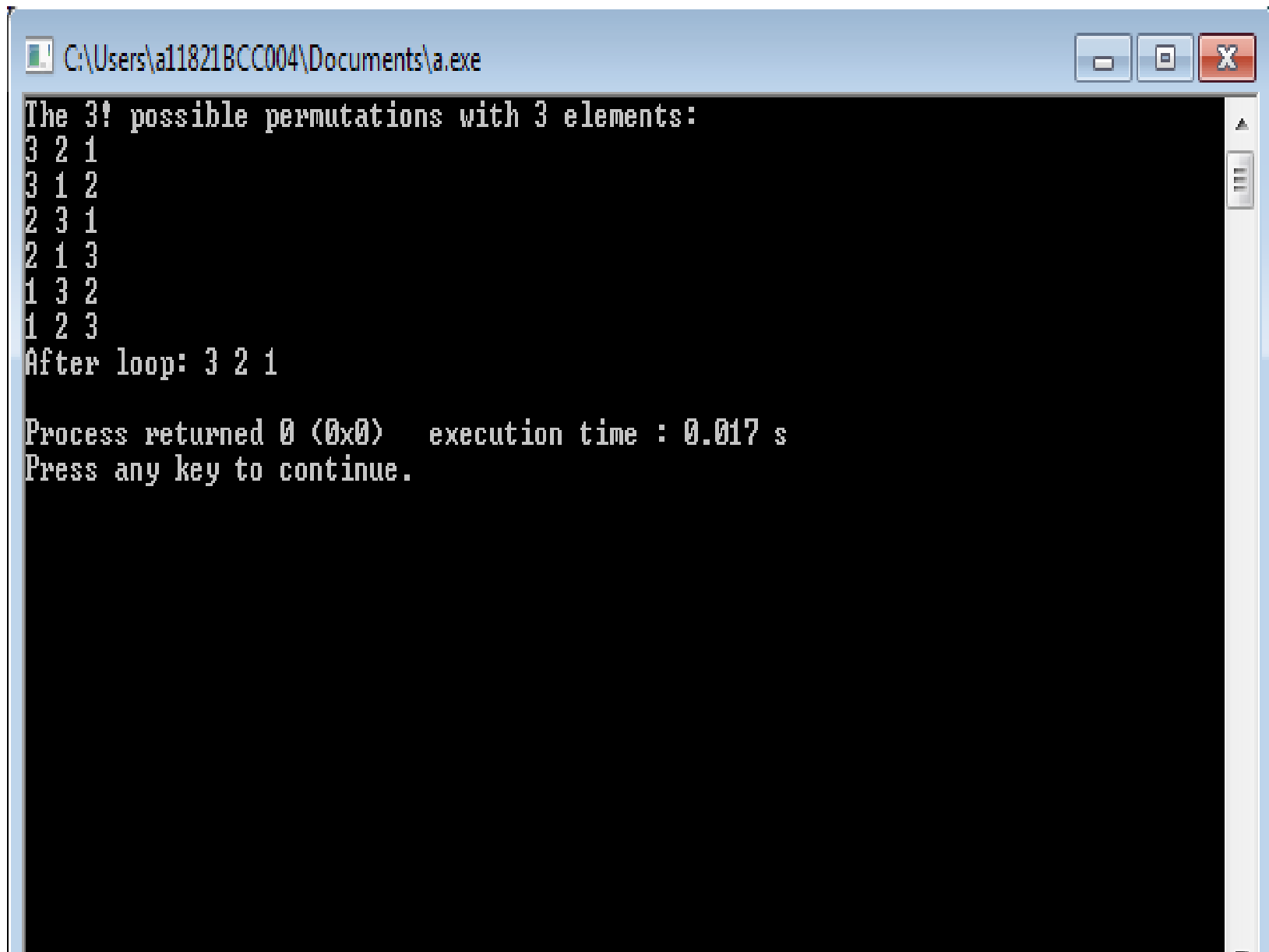
- Reorganiza os elementos no intervalo [first,last) na permutação ordenada lexicograficamente anterior, onde uma permutação é cada um dos possíveis  $N$  arranjos dos elementos do intervalo.



```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int main () {
5      int myints[] = {1,2,3};
6
7      sort (myints,myints+3);
8      reverse (myints,myints+3);
9
10     cout << "The 3! possible permutations with 3 elements:\n";
11     do {
12         cout << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';
13     } while ( prev_permutation(myints,myints+3) );
14
15     cout << "After loop: " << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';
16
17     return 0;
18 }
19
```



- Saída :



```
C:\Users\al1821BCC004\Documents\a.exe

The 3! possible permutations with 3 elements:
3 2 1
3 1 2
2 3 1
2 1 3
1 3 2
1 2 3
After loop: 3 2 1

Process returned 0 (0x0)   execution time : 0.017 s
Press any key to continue.
```

# IS\_PERMUTATION

- Compara e testa se os elementos no intervalo[first1,last1) é permutação de outro, e retorna true, se todos os elementos de ambos intervalos se corresponderem.

```
// is_permutation example
#include <iostream>      // std::cout
#include <algorithm>     // std::is_permutation
#include <array>         // std::array

int main () {
    std::array<int,5> foo = {1,2,3,4,5};
    std::array<int,5> bar = {3,1,4,5,2};

    if ( std::is_permutation (foo.begin(), foo.end(), bar.begin()) )
        std::cout << "foo and bar contain the same elements.\n";

    return 0;
}
```





Brent Kunkle