TRABALHO FINAL – TF

Pablo Vieira <pablo.vieira@edu.pucrs.br>
Prof. Dr. Rafael Garibotti <rafael.garibotti@pucrs.br> – Orientador

Pontifícia Universidade Católica do Rio Grande do Sul – Escola Politécnica – Curso de Engenharia de Computação
Av. Ipiranga, 6681 – Bairro Partenon – CEP 90619-900 – Porto Alegre – RS

17 de novembro de 2021

RESUMO

Este trabalho tem como objetivo utilizar na prática os conceitos e ferramentas de teste de software estudados durante o semestre.

Palavras-chave: Teste de Software; Teste de Unidade; Classes de equivalência, *Continuous Integration*.

1 INTRODUÇÃO

Algoritmo de ordenação, em ciência da computação, é um algoritmo que coloca os elementos de uma dada sequência em uma certa ordem. Em outras palavras efetua sua ordenação completa ou parcial. O objetivo da ordenação é facilitar a recuperação dos dados de uma lista.

2 SORT_ARRAY

A função a ser testada é a sort_array. Para fazer uso, basta possuir os arquivos sort.h e sort.c.

A função recebe parâmetros, como abaixo e retorna um ponteiro para o array ordenado:

```
/**
 * Method that receives a pointer to an array that will be sorted,
 * his size and the constant of the method that will be used
 * @param  array  Array to be sorted
 * @param  size   Size of the array
 * @param  method Sorting algorithm constant
 * @return        Pointer to the sorted array
 */
int* sort_array(int *array, int size, int method){

    switch(method){
```

3 ALGORITMOS DE ORDENAMENTO

O parâmetro *method* da função sort_array, é um inteiro que representa um dos seguintes algoritmos: selection sort, insertion sort, shell sort, quick sort, heap sort e merge sort.

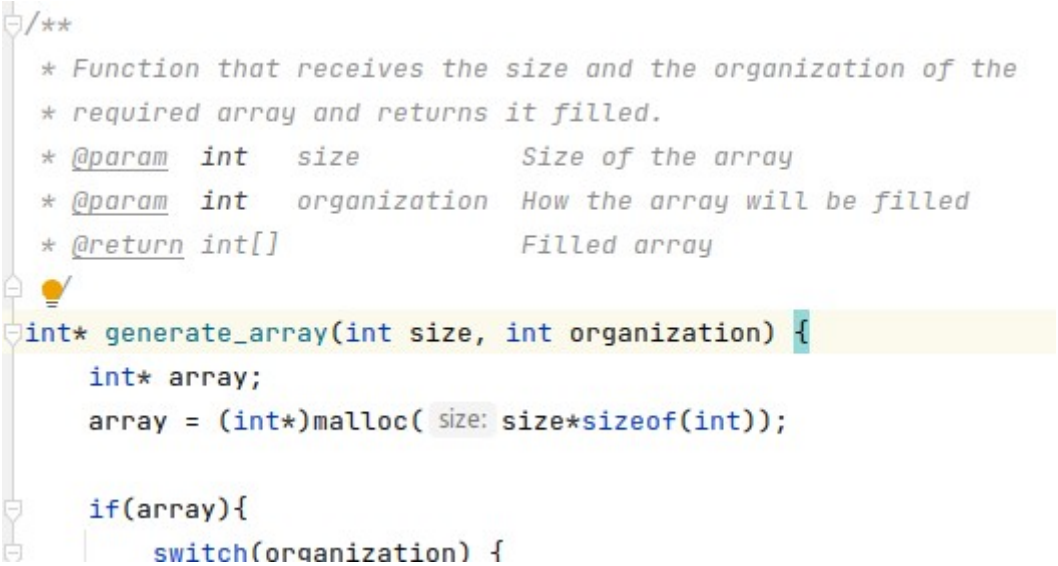
4 RECURSOS UTILIZADOS

Este projeto usa gcc como compilador, make para automatizar a compilação, e usa Travis CI como ferramenta de Continuous Integration. Além disso, vocês devem utilizar também o gcov como ferramenta de análise de cobertura de código, além das seguintes ferramentas voltas a teste: cppcheck, valgrind e sanitizer.

Também foi usada a ferramenta Unity para descrever os testes, já mostrada em aula.

5 GENERATE_ARRAY

A função generate_array, presente nos arquivos array.h e array.c também foi utilizada para ajudar na geração de *arrays*, que foram passados para a função alvo dos testes. Segue abaixo uma imagem com parte da função.



```
/**
 * Function that receives the size and the organization of the
 * required array and returns it filled.
 * @param  int   size          Size of the array
 * @param  int   organization  How the array will be filled
 * @return int[]               Filled array
 */
int* generate_array(int size, int organization) {
    int* array;
    array = (int*)malloc( size: size*sizeof(int));

    if(array){
        switch(organization) {
```

6 TESTES

Para tentar cobrir uma maior quantidade de testes foram utilizadas como entrada para a função sort_array 06 (seis) tipos de preenchimento de array de entrada. E 06 (seis) tipos de métodos de ordenamento.

A tabela abaixo apenas mostra os tipos de entrada. As linhas não estão associadas, porque todos foram cruzados com todos.

Fills arrays	*Sorting methods*
Arrays with totally random elements	*Selection sort*
Arrays already ordered	*Insertion sort*
Arrays ordered in descending order	*Shell sort*
Arrays 90% ordered	*Quick sort*
Array de números negativos	*Heap sort*
Array de 0 (zeros)	*Merge sort*

Código com 24 possibilidades:

```
TEST(SortArray, TestSortArray7)
{
    n = 10;

    for( int state = 0; state <= 3; state++){ // state: RANDOM_ORDER, ASCENDING_ORDER, DESCENDING_ORDER, ALMOST_ORDERED
        // Create the vector with the specified size and situation
        int *vet = generate_array(n, state);

        for(int algorithm = 0; algorithm < 7; algorithm++) { // algorithm: SELECTION, INSERTION, SHELL, QUICK, HEAP, MERGE
            sort_array(vet, n, algorithm);
        }
    }
}
```

Número do Teste	Nome do Teste	Casos de Teste
1	TestSortArray1	{40, 10, 100, 90, 20, 25},{10, 20, 25, 40, 90, 10}
2	TestSortArray2	{6},{6}
3	TestSortArray3	{6},{6}
4	TestSortArray4	{3 4 4 -1 2 -1 4 3 6 1},{-1 -1 1 2 3 3 4 4 4 6}
5	TestSortArray5	{-1 0 1 2 3 4 5 6 7 6},{-1 0 1 2 3 4 5 6 6 7}
6	TestSortArray6	{8 7 6 5 4 3 2 1 0 -1},{-1 0 1 2 3 4 5 6 7 8}
7	TestSortArray7	{-1 0 1 2 3 4 5 6 7 9},{-1 0 1 2 3 4 5 6 7 9}
8	TestSortArray8	{0 0 0 0 0},{0 0 0 0 0}
9	TestSortArray9	{-1 -5 -2 -4 -3},{-5 -4 -3 -2 -1}