INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS SÃO JOÃO EVANGELISTA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

JOÃO VITOR MENDES CAMPOS
GABRIEL KÁICON BATISTA HILÁRIO

TRABALHO PRÁTICO II

SÃO JOÃO EVANGELISTA
2023

JOÃO VITOR MENDES CAMPOS
GABRIEL KÁICON BATISTA HILÁRIO

SISTEMA DE RH DE UMA EMPRESA

SÃO JOÃO EVANGELISTA
2023

SUMÁRIO

1. INTRODUÇÃO

Este trabalho prático foi documentado para que seja avaliado em conjunto com os códigos na linguagem C/C++, exigido pelo docente Eduardo Augusto da Costa Trindade, dentro da disciplina de Algoritmos e Estruturas de Dados II, ministrada pelo mesmo. Porém a documentação tem cunho expositivo, onde é descrito as funcionalidades do programa, com testes, e desenvolvimento de novas linhas de raciocínio lógico para realização do trabalho prático.

1.1. Objetivo Geral

Este trabalho tem como objetivo geral apresentar na prática os conhecimentos adquiridos nas aulas de Algoritmos e Estruturas de Dados II, a respeito das Árvores Binária de Busca (BST), utilizando a linguagem C++ para escrita dos códigos.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Fixar conceitos sobre manipulação de árvores binárias.
- Realizar a implementação de árvore binária e suas TAD'S
- Compreender os métodos de inserção, busca e remoção em uma árvore.
- Estimular o raciocínio lógico para a resolução de problemas.

1.3. Justificativa

Ao iniciar os estudos de Algoritmos e Estruturas de Dados II, vemos os conteúdos Análise de Complexidade, Custo da função, Big O, Recursividade, Divisão e conquista Métodos de Ordenação, Árvore Binária de Busca como uma nova Estrutura de Dados, até então desconhecida, porém o que nos interessa nesse momento é a Árvore Binária de Busca (BST).

O objetivo principal de uma BST é permitir uma busca eficiente de dados, a recuperação rápida e eficiente de elementos com base em uma chave de pesquisa, geralmente uma chave numérica ou alfabética. Essa estrutura de dados é ótima para desenvolver algoritmos com bom desempenho simultâneo para a busca e inserção, tendo em vista que a mesma utiliza recursividade. Além disso, as árvores binárias de busca podem ser usadas para realizar outras operações como: remoção, mínimo, máximo, antecessor e sucessor. O trabalho foi exigido para que seja possível desenvolver o raciocínio lógico quanto a aplicação dessa estrutura, e com o bônus de novamente aplicá-la em conjunto.

2. DESENVOLVIMENTO

Nesta seção do documento é apresentado, os conceitos aprendidos e o desenvolvimento do trabalho em si, na linguagem C++.

2.1. Árvore Binária de Busca

A propriedade fundamental que define uma Árvore Binária de Busca é que, para cada nó, todos os valores nos filhos à esquerda do nó são menores que o valor do próprio nó, e todos os valores nos filhos à direita do nó são maiores. Isso permite uma busca binária eficiente, em que a busca é iniciada a partir do nó raiz, e o valor da chave é comparado com o valor do nó atual. Dependendo do resultado da comparação, a busca continua na sub-árvore à esquerda ou sub-árvore à direita.

2.2. Implementação

A modularização feita foi a main.cpp, BST.hpp, BST.cpp e um main.exe. Na parte de implementação foram utilizadas as funções fornecidas pelo professor.

Apêndice A - Nele é apresentada a função (Cria No) o objetivo dessa função é criar um nó na árvore, essa função é do tipo No e ao final de sua execução retornará um novo nó.

Apêndice B - Nele é apresentada a função (Insere No) o objetivo dessa função é inserir um nó na árvore, essa função é do tipo No, ela verifica se já existe uma raiz, se a raiz existe ela continua a executar normalmente, mas se a raiz não existir a função criará essa raiz, seguindo a função verifica se a o objeto a ser inserido é maior que a raiz se assim ocorrer esse objeto será inserido à direita da raiz, mas se o objeto for menor que a raiz será inserida à esquerda da mesma, após o término da execução é retornado a raiz.

Apêndice C - Nele é apresentada a função (Remove No) o objetivo é remover um nó, essa função é do tipo No, ela verifica a existência da raiz se ela existir o programa continua a ser executado, mas se a raiz não existir será retornada a raiz, seguindo a verificado se o objeto a ser removido é menor que a raiz assim removendo o objeto da parte esquerda da árvore, mas se o objeto a ser removido é maior que a raiz o objeto será removido da parte direita da árvore, ao final será retornado a raiz.

Apêndice D - Nele é apresentada a função (Cria Pessoa) o objetivo é criar o cadastro da pessoa, essa função é do tipo Pessoa, nela o usuário vai passar os dados da pessoa a serem cadastrada, alterando todos os campos da mesma ao final será retornado um “objeto” pessoa para ser inserida na árvore.

Apêndice E - Nele é apresentada a função (Imprime Item) o objetivo é imprimir os dados de uma pessoa já cadastrada, essa função é do tipo Void, nela serão imprimidos os

dados de uma pessoa, para fazermos uso do reaproveitamento de código, já que sempre vamos pedir para imprimir em diferentes funções.

Apêndice F - Nele é apresentada a função (Busca Matrícula) o objetivo é buscar por meio da matrícula uma pessoa, essa função é do tipo No, nela o usuário passa o número de matrícula da pessoa desejada, verifica- se a raiz da árvore existe ou se o número de matrícula é foi encontrado, se assim ocorrer será imprimindo os dados da pessoa e retornado a raiz, mas se isso não ocorrer, será verificado se a número de matrícula escolhido pelo usuário é menor que os números da matrícula contidos na árvore, se assim acontecer a função será executada novamente só que agora levando em conta a esquerda e direita da raiz como parâmetros.

Apêndice G - Nele é apresentada a função (Busca CPF) o objetivo é buscar por meio do CPF da pessoa, essa função é do tipo No, nela o usuário passa o CPF da pessoa desejada, verifica- se a raiz da árvore existe ou se o CPF foi encontrado, se assim ocorrer será imprimindo os dados da pessoa e retornado a raiz, mas se isso não ocorrer, será verificado se o CPF escolhido pelo usuário é diferente dos CPFs contidos na árvore, se assim acontecer a função será executada novamente só que agora levando em conta a esquerda e direita da raiz como parâmetros.

Apêndice H - Nele é apresentada a função (Busca Nome) o objetivo é buscar por meio do nome da pessoa, essa função é do tipo No, nela o usuário passa o nome da pessoa desejada, verifica- se a raiz da árvore existe ou se o nome foi encontrado, se assim ocorrer será imprimindo os dados da pessoa e retornado a raiz, mas se isso não ocorrer, será verificado se o nome escolhido pelo usuário é diferente dos nomes contidos na árvore, se assim acontecer a função será executada novamente só que agora levando em conta a esquerda e direita da raiz como parâmetros.

Apêndice I - Nele é apresentada a função (Remove Matrícula) o objetivo é remover o cadastro de uma pessoa por meio do número de matrícula da pessoa que o usuário escolheu, essa função é do tipo Void, nela o usuário informa o número da matrícula da pessoa a ser removida, a função (Remove No) recebe o número de matrícula e remove o cadastro da árvore.

Apêndice J - Nele é apresentada a função (Imprime Ordem) o objetivo é imprimir a árvore em ordem, que seria a impressão começando da sub-árvore à esquerda, raiz e depois a sub-árvore à direita. Essa função é do tipo Void.

Apêndice K - Nele é apresentada a função (Imprime Pré-Ordem) o objetivo é imprimir a árvore em pré-ordem, que seria a impressão partindo da Raiz, depois as árvores e/ou sub-árvores à esquerda e depois e/ou sub-árvores à direita. Essa função é do tipo Void.

Apêndice L - Nele é apresentada a função (Imprime Pós-Ordem) o objetivo é imprimir a árvore em pós-ordem, isso seria a impressão começando a partir da sub-árvore esquerda, depois a sub-árvore direita, e por fim a raiz. Essa função é do tipo Void.

Apêndice M - Nele é apresentada a função (Busca) o objetivo é buscar os dados de uma pessoa na árvore, essa função é do tipo Void, nela o usuário escolhe se a busca será por CPF, Matrícula ou Nome, se selecionado CPF será executado a função (Busca CPF), se selecionado Nome será executado a função (Busca Nome) e se selecionado Matrícula será executado a função (Busca Matrícula), assim retornando os dados da pessoa.

Apêndice N - Nele é apresentada a função (Imprime) o objetivo é imprimir a árvore a partir do modo escolhido pelo usuário, essa função é do tipo Void, nela o usuário escolhe se a impressão da árvore será em Ordem, Pré-Ordem ou Pós-ordem, após a escolha a árvore será impressa segundo o modo selecionado.

Apêndice O - Nele é apresentado a função (Menu Busca) o objetivo é imprimir um menu de busca, essa função é do tipo Int, neste menu estão os modos de busca que podem ser escolhidos, após a escolha do modo de busca será retornada a opção escolhida.

Apêndice P - Nele é apresentado a função (Menu) o objetivo é imprimir o menu principal, essa função é do tipo Void, será apresentado na tela o menu com as opções 1- Cadastrar Funcionário, 2- Buscar Funcionário, 3- Remover Funcionário, 4- Imprimir, 5- Sair e opção:.

Apêndice Q - Nele é apresentada a função (Menu Imprimir) o objetivo é imprimir um menu dos modos de impressão, essa função é do tipo Int, nela é apresentado os modos de impressão, é pedido que o usuário escolha qual modo de impressão ele deseja.

Apêndice R - Nele é apresentada a TAD escolhida para a execução do trabalho.

3. CONCLUSÃO

Ao longo do trabalho tivemos várias ideias de como realizar a implementação do trabalho, já que a Árvore Binária de Busca pode ser feita de várias formas. Utilizamos os códigos padrões da criação de árvores, e damos um foco maior nas funções de inserção, remoção, impressão, e busca por diferentes tipos de dados, também fizemos uso dos slides disponibilizados pelo professor para sanar dúvidas a respeito, e usuários externos para tirar dúvidas sobre a criação e uso de novas funções. Creio que o desenvolvimento foi bem satisfatório nesse trabalho, estávamos bem receosos com o código, depois vimos a simplicidade no desenvolvimento. Nós desenvolvemos novas formas de raciocínio e tivemos que abstrair e diminuir muitas coisas como de costume, para que assim o trabalho ficasse leve, e modularizado, sem ser muito extenso. Tivemos que buscar ajuda externa e conhecimento externo para realização do trabalho, mas mesmo assim conseguimos atingir nossos objetivos estipulados no início do trabalho, explicar bem o que estava fazendo dentro de cada função, apresentar conhecimentos em Árvore Binária de Busca, e com confiança aprimorar nossos conhecimentos. Estamos felizes e satisfeitos com o resultado, e ele atendeu às nossas expectativas além do que era esperado.

4. REFERÊNCIAS

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados II – ÁRVORE BINÁRIA DE BUSCA (BST). 2023. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/sje-presencial/pluginfile.php/158749/mod_resource/content/1/Aula 13 - Árvore Binária de Busca (BST).pdf . Acesso em: 08 de maio de 2023.

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados II – ÁRVORES. 2023. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/sje-presencial/pluginfile.php/158748/mod_resource/content/1/Aula%2012%20-%20%C3%81rvores.pdf . Acesso em: 08 de maio de 2023.

Link no github https://github.com/gKaicon/Arquivos_C_and_C-withClasses/tree/main/AEDs%20II%2FTP_%C3%81rvoreBuscaBin%C3%A1ria

5. APÊNDICES

APÊNDICE A – (Cria No)

```
No *criarNo(Pessoa pessoa)
{
    No *novoNo = new No();
    novoNo->pessoa = pessoa;
    novoNo->esquerda = nullptr;
    novoNo->direita = nullptr;
    return novoNo;
}
```

APÊNDICE B – (Insere No)

```
No *inserirNo(No *raiz, Pessoa pessoa)
{
    if (raiz == nullptr)
    {
        raiz = criarNo(pessoa);
    }
    else if (pessoa.matricula < raiz->pessoa.matricula)
    {
        raiz->esquerda = inserirNo(raiz->esquerda, pessoa);
    }
    else
    {
        raiz->direita = inserirNo(raiz->direita, pessoa);
    }
    return raiz;
}
```

APÊNDICE C – (Removê No)

```
No *removerNo(No *raiz, int matricula)
{
    if (raiz == nullptr)
    {
        return raiz;
    }
    else if (matricula < raiz->pessoa.matricula)
    {
        raiz->esquerda = removerNo(raiz->esquerda, matricula);
    }
    else if (matricula > raiz->pessoa.matricula)
    {
        raiz->direita = removerNo(raiz->direita, matricula);
    }
    else
    {
        // Caso 1: Nó não possui filhos
        if (raiz->esquerda == nullptr && raiz->direita == nullptr)
        {
            delete raiz;
            cout << "Matricula removida.";
            raiz = nullptr;
        }
        // Caso 2: Nó possui apenas um filho
        else if (raiz->esquerda == nullptr)
        {
            No *temp = raiz;
            raiz = raiz->direita;
            delete temp;
            cout << "Matricula removida.";
        }
        else if (raiz->direita == nullptr)
        {
            No *temp = raiz;
            raiz = raiz->esquerda;
            delete temp;
            cout << "Matricula removida.";
        }
        // Caso 3: Nó possui dois filhos
        else
        {
            No *temp = raiz->direita;
            while (temp->esquerda != nullptr)
            {
                temp = temp->esquerda;
            }
            raiz->pessoa = temp->pessoa;
            raiz->direita = removerNo(raiz->direita, temp->pessoa.matricula);
            cout << "Matricula removida.";
        }
    }
    return raiz;
}
```

APÊNDICE D – (Cria Pessoa)

```
Pessoa criarPessoa()
{
    Pessoa pessoa;
    cout << "Matricula: ";
    cin >> pessoa.matricula;
    cin.ignore();
    cout << "CPF: ";
    getline(cin, pessoa.cpf);
    cout << "Nome: ";
    getline(cin, pessoa.nome);
    cout << "Data de nascimento - Dia: ";
    cin >> pessoa.data_nascimento.dia;
    cout << "Data de nascimento - Mês: ";
    cin >> pessoa.data_nascimento.mes;
    cout << "Data de nascimento - Ano: ";
    cin >> pessoa.data_nascimento.ano;
    cin.ignore();
    cout << "Endereço - Rua: ";
    getline(cin, pessoa.endereco.logradouro);
    cout << "Endereço - Numero: ";
    cin >> pessoa.endereco.numero;
    cin.ignore();
    cout << "Endereço - Bairro: ";
    getline(cin, pessoa.endereco.bairro);
    cout << "Endereço - CEP: ";
    getline(cin, pessoa.endereco.cep);
    cout << "Endereço - Cidade: ";
    getline(cin, pessoa.endereco.cidade);
    cout << "Endereço - Estado: ";
    getline(cin, pessoa.endereco.estado);
    cout << "Telefone: ";
    getline(cin, pessoa.telefone);
    cout << "Cargo: ";
    getline(cin, pessoa.cargo);
    return pessoa;
}
```

APÊNDICE E – (Imprime Item)

```
void imprimeItem(Pessoa pessoa)
{
    cout << "\nMatricula: " << pessoa.matricula
         << "\nCPF: " << pessoa.cpf
         << "\nNome: " << pessoa.nome
         << "\nData de nascimento - Dia: " << pessoa.data_nascimento.dia
         << "\nData de nascimento - Mês: " << pessoa.data_nascimento.mes
         << "\nData de nascimento - Ano: " << pessoa.data_nascimento.ano
         << "\nEndereço - Rua: " << pessoa.endereco.logradouro
         << "\nEndereço - Numero: " << pessoa.endereco.numero
         << "\nEndereço - Bairro: " << pessoa.endereco.bairro
         << "\nEndereço - CEP: " << pessoa.endereco.cep
         << "\nEndereço - Cidade: " << pessoa.endereco.cidade
         << "\nEndereço - Estado: " << pessoa.endereco.estado
         << "\nTelefone: " << pessoa.telefone
         << "\nCargo: " << pessoa.cargo << endl << endl;
}
```

APÊNDICE F – (Busca Matrícula)

```
No *buscarMatricula(No *raiz, int matricula)
{
    if (raiz == nullptr || raiz->pessoa.matricula == matricula)
    {
        imprimeItem(raiz->pessoa);
        return raiz;
    }
    if (raiz->pessoa.matricula < matricula)
        return buscarMatricula(raiz->direita, matricula);
    return buscarMatricula(raiz->esquerda, matricula);
}
```

APÊNDICE G - (Busca CPF)

```
No *buscarCPF(No *raiz, string cpf)
{
    if (raiz == nullptr || raiz->pessoa.cpf == cpf)
    {
        imprimeItem(raiz->pessoa);
        return raiz;
    }

    if (raiz->pessoa.cpf != cpf)
        return buscarCPF(raiz->direita, cpf);
    return buscarCPF(raiz->esquerda, cpf);
}
```

APÊNDICE H - (Busca Nome)

```
No *buscarNome(No *raiz, string nome)
{
    if (raiz == nullptr || (raiz->pessoa.nome == nome))
    {
        imprimeItem(raiz->pessoa);
        return raiz;
    }
    if (nome != raiz->pessoa.nome)
        return buscarNome(raiz->direita, nome);
    return buscarNome(raiz->esquerda, nome);
}
```

APÊNDICE I - (Remove Matrícula)

```
void removeMatricula(No *raiz){
    int matricula;
    cout << "Digite a matricula a ser Excluída(caso não se lembre, digite qualquer valor negativo e confira): ";
    cin >> matricula;
    removerNo(raiz, matricula);
    cout << endl;
    system("PAUSE");
}
```

APÊNDICE J - (Imprime Ordem)

```
void imprimeEmOrdem(No *raiz)
{
    if (raiz != nullptr)
    {
        imprimeEmOrdem(raiz->esquerda);
        imprimeItem(raiz->pessoa);
        imprimeEmOrdem(raiz->direita);
    }
}
```

APÊNDICE K - (Imprime Pré-Ordem)

```
void imprimePreOrdem(No *raiz)
{
    if (raiz != nullptr)
    {
        imprimeItem(raiz->pessoa);
        imprimePreOrdem(raiz->esquerda);
        imprimePreOrdem(raiz->direita);
    }
}
```

APÊNDICE L - (Imprime Pós-Ordem)

```
void imprimePosOrdem(No *raiz)
{
    if (raiz != nullptr)
    {
        imprimePosOrdem(raiz->esquerda);
        imprimePosOrdem(raiz->direita);
        imprimeItem(raiz->pessoa);
    }
}
```

APÊNDICE M - (Busca)

```
void Busca(No *raiz, int op){
    int matricula;
    string nome, cpf;

    if(op == 1){
        cout << "CPF: ";
        getline(cin, cpf);
        buscarCPF(raiz, cpf);
    }
    if (op == 2)
    {
        cout << "Nome: ";
        getline(cin, nome);
        buscarNome(raiz, nome);
    }
    if (op == 3)
    {
        cout << "Matricula: ";
        cin >> matricula;
        cin.ignore();
        buscarMatricula(raiz, matricula);
    }
    system("PAUSE");
}
```

APÊNDICE N - (Imprime)

```
void imprime(No *raiz, int op){
    if(op == 1){
        imprimeEmOrdem(raiz);
    }
    if (op == 2){
        imprimePreOrdem(raiz);
    }
    if (op == 3){
        imprimePosOrdem(raiz);
    }
}
```

APÊNDICE O - (Menu Busca)

```
int menubusca(){
    int opB;
    cout << "  \n        ********************************************************************";
    cout << "  \n        ********************************************************************";
    cout << "  \n        **                                                                **";
    cout << "  \n        **                  ESCOLHA O MODO DE BUSCA                       **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ********************************************************************";
    cout << "  \n        ********************************************************************";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 1 - BUSCA POR CPF                                             **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 2 - BUSCA POR NOME                                            **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 3 - BUSCA POR MATRICULA                                       **";
    cout << "  \n        **                                                                **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ********************************************************************";
    cout << "  \n        ********************************************************************\n\n";
    cout << "Opção: ";
    cin >> opB;
    cin.ignore();
    return opB;
}
```

APÊNDICE P - (Menu)

```
void menu(){
    cout << "  \n        ********************************************************************";
    cout << "  \n        ********************************************************************";
    cout << "  \n        **                                                                **";
    cout << "  \n        **                      MENU-PRINCIPAL                            **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ********************************************************************";
    cout << "  \n        ********************************************************************";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 1 - CADASTRAR FUNCIONÁRIO                                     **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 2 - BUSCAR FUNCIONÁRIO                                        **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 3 - REMOVER FUNCIONÁRIO                                       **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 4 - IMPRIMIR                                                  **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ** 5 - SAIR                                                      **";
    cout << "  \n        **                                                                **";
    cout << "  \n        ********************************************************************";
    cout << "  \n        ********************************************************************\n\n";
    cout << "Opção: ";
}
```

APÊNDICE Q - (Menu Imprimir)

```
int menuimprimir(){
    int opI;
    cout << "  \n        ****************************************************************";
    cout << "  \n        ****************************************************************";
    cout << "  \n        **                                                            **";
    cout << "  \n        **               ESCOLHA O MODO DE IMPRESSÃO                  **";
    cout << "  \n        **                                                            **";
    cout << "  \n        ****************************************************************";
    cout << "  \n        ****************************************************************";
    cout << "  \n        **                                                            **";
    cout << "  \n        ** 1 - IMPRESSÃO EM ORDEM                                     **";
    cout << "  \n        **                                                            **";
    cout << "  \n        ** 2 - IMPRESSÃO PRÉ-ORDEM                                    **";
    cout << "  \n        **                                                            **";
    cout << "  \n        ** 3 - IMPRESSÃO PÓS-ORDEM                                    **";
    cout << "  \n        **                                                            **";
    cout << "  \n        ****************************************************************";
    cout << "  \n        ****************************************************************\n\n";
    cout << "Opção: ";
    cin >> opI;
    if (opI == 1) cout << "\nImpressão em ordem.\n\n";
    if (opI == 2) cout << "\nImpressão pré-ordem.\n\n";
    if (opI == 3) cout << "\nImpressão pós-ordem.\n\n";

    return opI;
}
```

APÊNDICE R - (TAD)

```
typedef struct Data
{
    unsigned int dia;
    unsigned int mes;
    unsigned int ano;
};

typedef struct Endereco
{
    string logradouro;
    int numero;
    string bairro;
    string cidade;
    string estado;
    string cep;
};

typedef struct Pessoa
{
    int matricula;
    string cpf;
    string nome;
    string cargo;
    string telefone;
    Data data_nascimento;
    Endereco endereco;
};
```

```
struct No
{
    Pessoa pessoa;
    No *esquerda;
    No *direita;
};

No *criarNo(Pessoa pessoa);
No *inserirNo(No *raiz, Pessoa pessoa);
No *removerNo(No *raiz, int matricula);
Pessoa criarPessoa();
void imprimeItem(Pessoa pessoa);
No *buscarMatricula(No *raiz, int matricula);
No *buscarCPF(No *raiz, string cpf);
No *buscarNome(No *raiz, string nome);
No *removerPorMatricula(No *raiz, int matricula);
void removeMatricula(No *raiz);
void imprimeEmOrdem(No *raiz);
void imprimePreOrdem(No *raiz);
void imprimePosOrdem(No *raiz);
void Busca(No *raiz, int op);
void imprime(No *raiz, int op);
int menubusca();
void menu() ;
int menuimprimir();
```