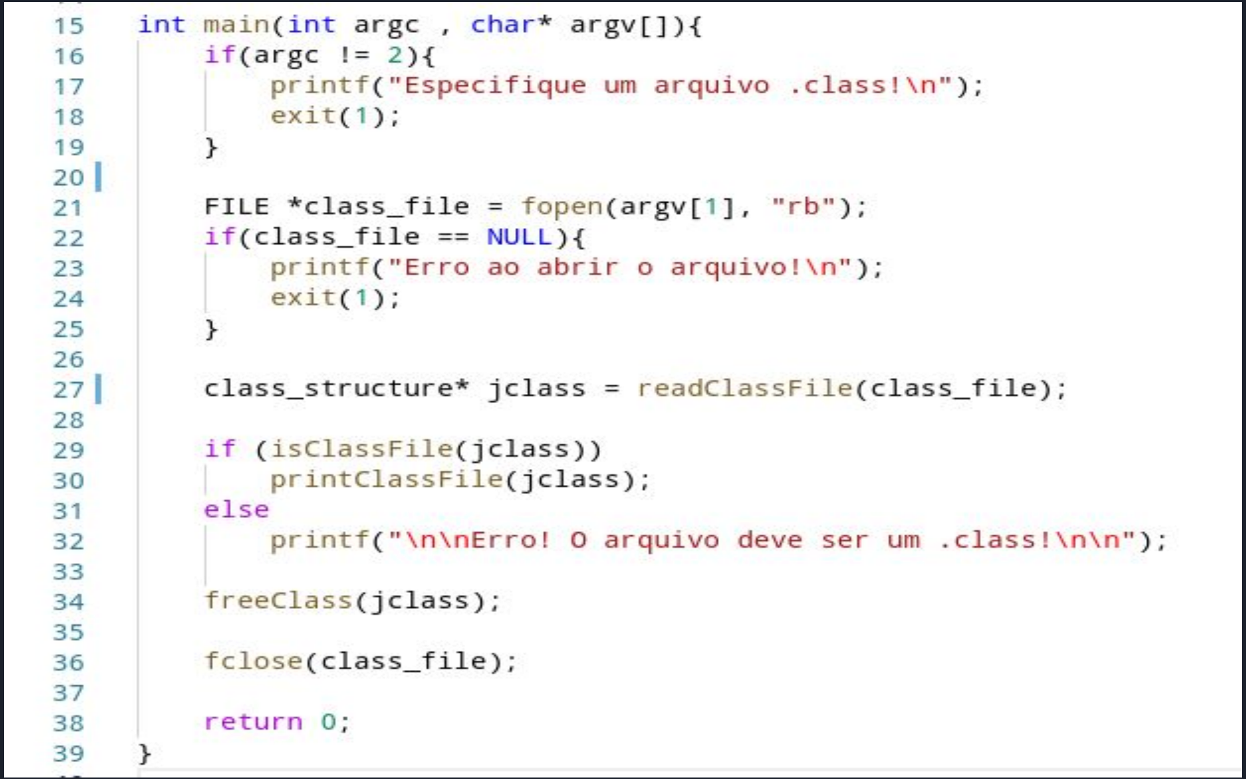
Leitor e Exibidor de .class

main



```
int main(int argc , char* argv[]){
    if(argc != 2){
        printf("Especifique um arquivo .class!\n");
        exit(1);
    }

    FILE *class_file = fopen(argv[1], "rb");
    if(class_file == NULL){
        printf("Erro ao abrir o arquivo!\n");
        exit(1);
    }

    class_structure* jclass = readClassFile(class_file);

    if (isClassFile(jclass))
        printClassFile(jclass);
    else
        printf("\n\nErro! O arquivo deve ser um .class!\n\n");

    freeClass(jclass);

    fclose(class_file);

    return 0;
}
```

j8_class_reader.c

- Verificação de validez do *.class*
- Leitura do .class
    - Constant_pool
    - Interface
    - Methods
    - Fields
    - Attributes
- Desalocação de memória

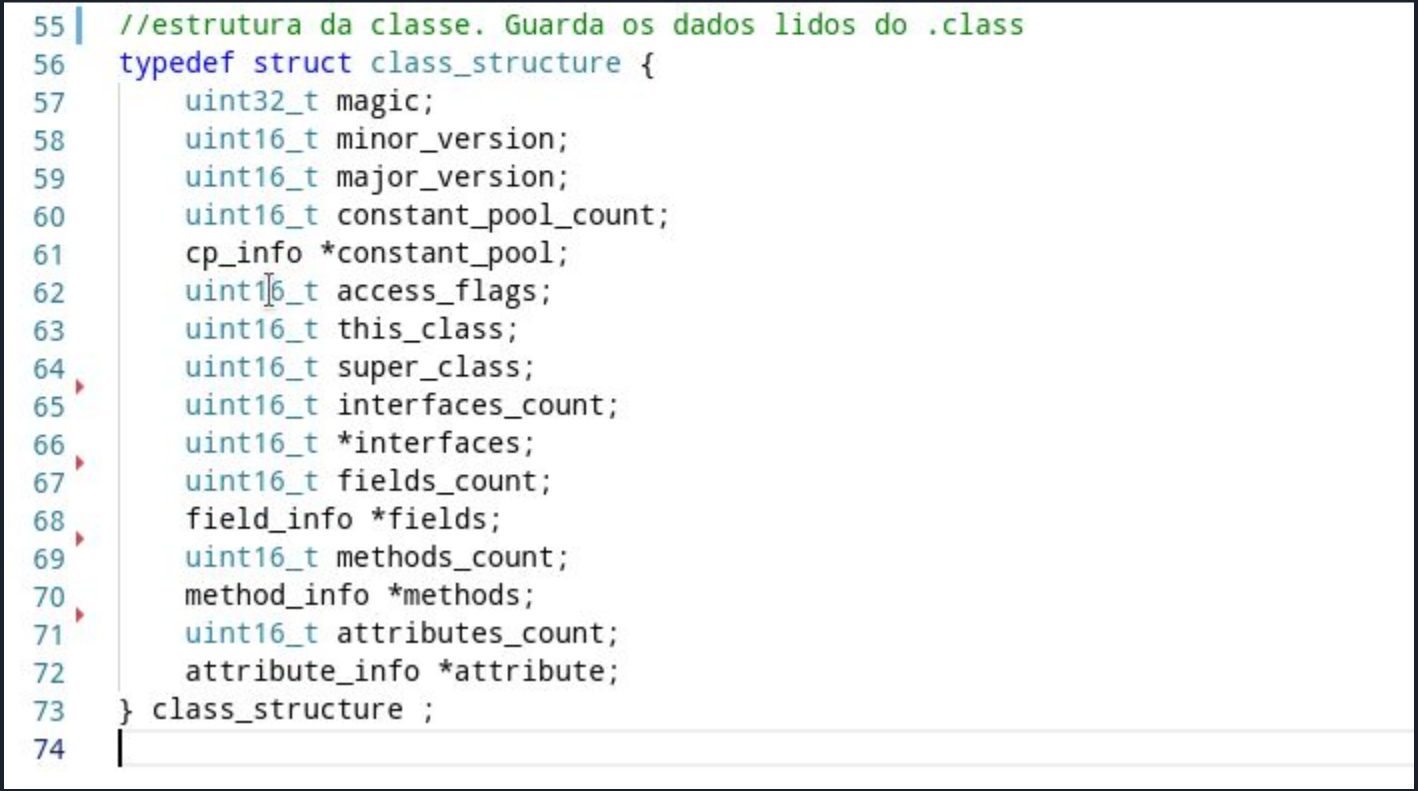
j8_class_displayer.c

- Mostra todas os campos do *.class* no terminal
- Para referências de valores do *constant_pool*:
  - # <número>   -> para valor da referência
  - // <conteúdo>   -> para o valor legível
- Para *Acess Flags* e atributos do tipo *Code*:
  - Mostra o Hexadecimal
  - Mostra o valor legível

read_utils.c e opcode.c

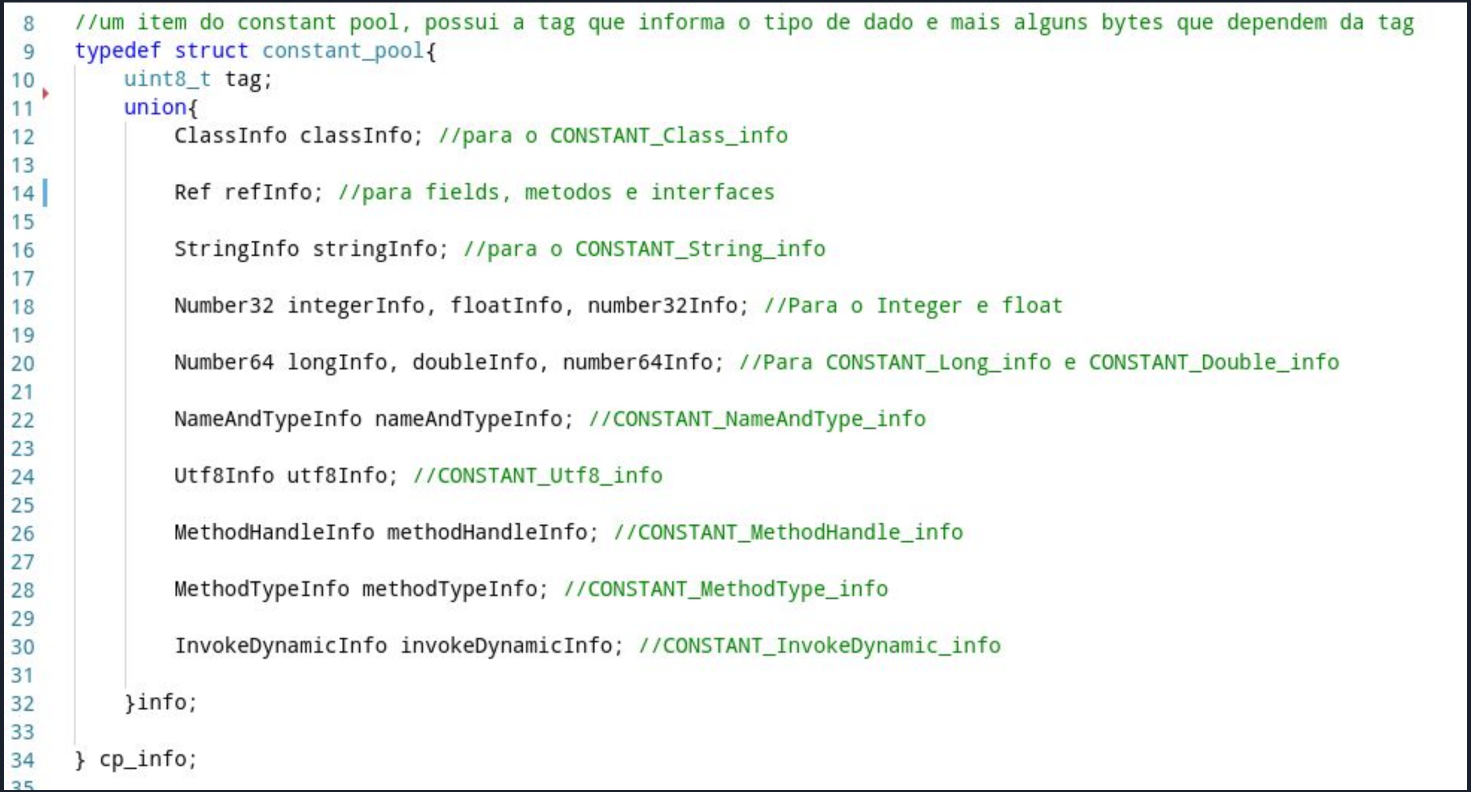
- read_utils
  - Auxilia na leitura do binário *.class*
  - Converte de bytecode de *big endian* para *little endian*
- opcode
  - Auxilia na legibilidade de cada *opcode*

Estrutura do class file



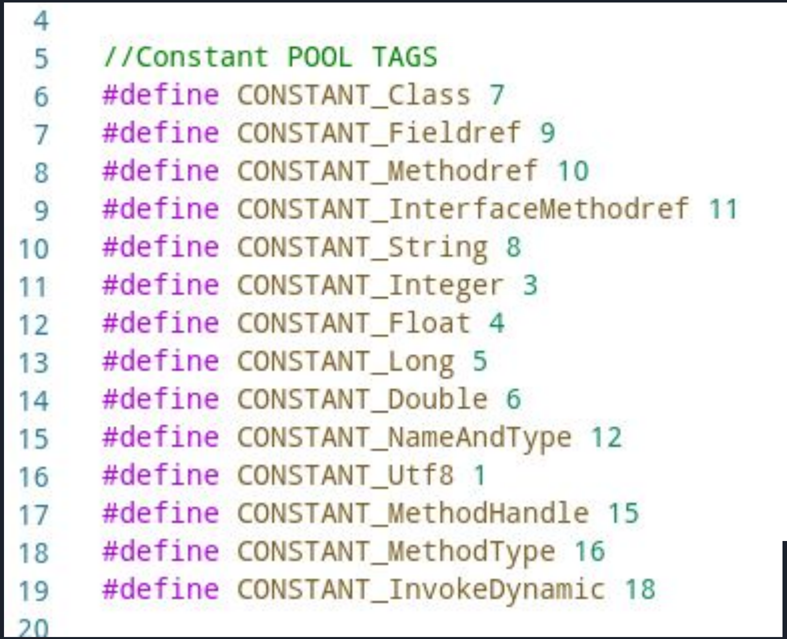
```
//estrutura da classe. Guarda os dados lidos do .class
typedef struct class_structure {
    uint32_t magic;
    uint16_t minor_version;
    uint16_t major_version;
    uint16_t constant_pool_count;
    cp_info *constant_pool;
    uint16_t access_flags;
    uint16_t this_class;
    uint16_t super_class;
    uint16_t interfaces_count;
    uint16_t *interfaces;
    uint16_t fields_count;
    field_info *fields;
    uint16_t methods_count;
    method_info *methods;
    uint16_t attributes_count;
    attribute_info *attribute;
} class_structure ;
```

Constant Pool

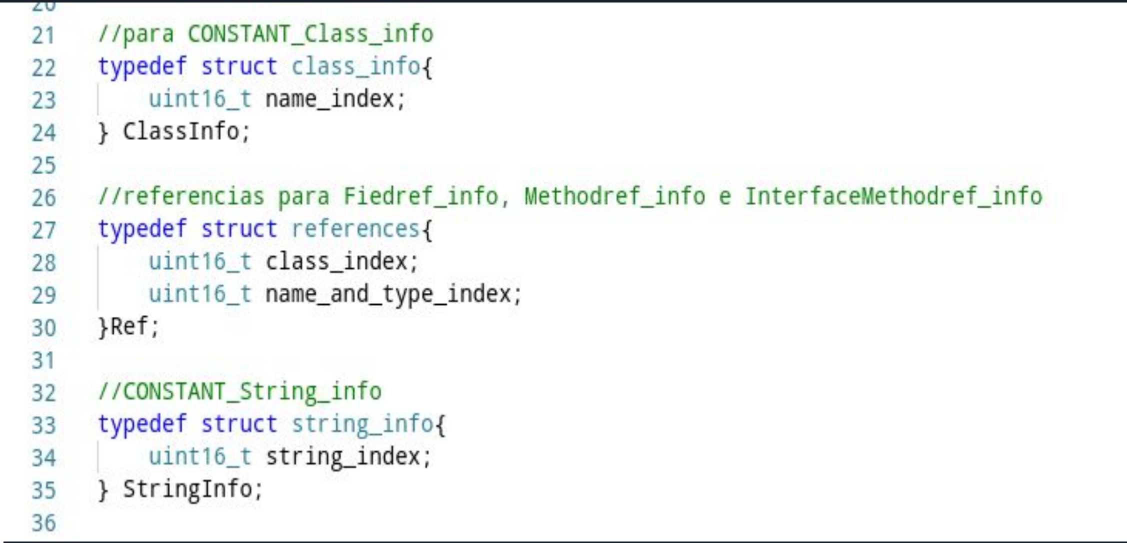


```c
//um item do constant pool, possui a tag que informa o tipo de dado e mais alguns bytes que dependem da tag
typedef struct constant_pool{
    uint8_t tag;
    union{
        ClassInfo classInfo; //para o CONSTANT_Class_info

        Ref refInfo; //para fields, metodos e interfaces

        StringInfo stringInfo; //para o CONSTANT_String_info

        Number32 integerInfo, floatInfo, number32Info; //Para o Integer e float

        Number64 longInfo, doubleInfo, number64Info; //Para CONSTANT_Long_info e CONSTANT_Double_info

        NameAndTypeInfo nameAndTypeInfo; //CONSTANT_NameAndType_info

        Utf8Info utf8Info; //CONSTANT_Utf8_info

        MethodHandleInfo methodHandleInfo; //CONSTANT_MethodHandle_info

        MethodTypeInfo methodTypeInfo; //CONSTANT_MethodType_info

        InvokeDynamicInfo invokeDynamicInfo; //CONSTANT_InvokeDynamic_info

    }info;

} cp_info;
```

j8_constant_pool.h

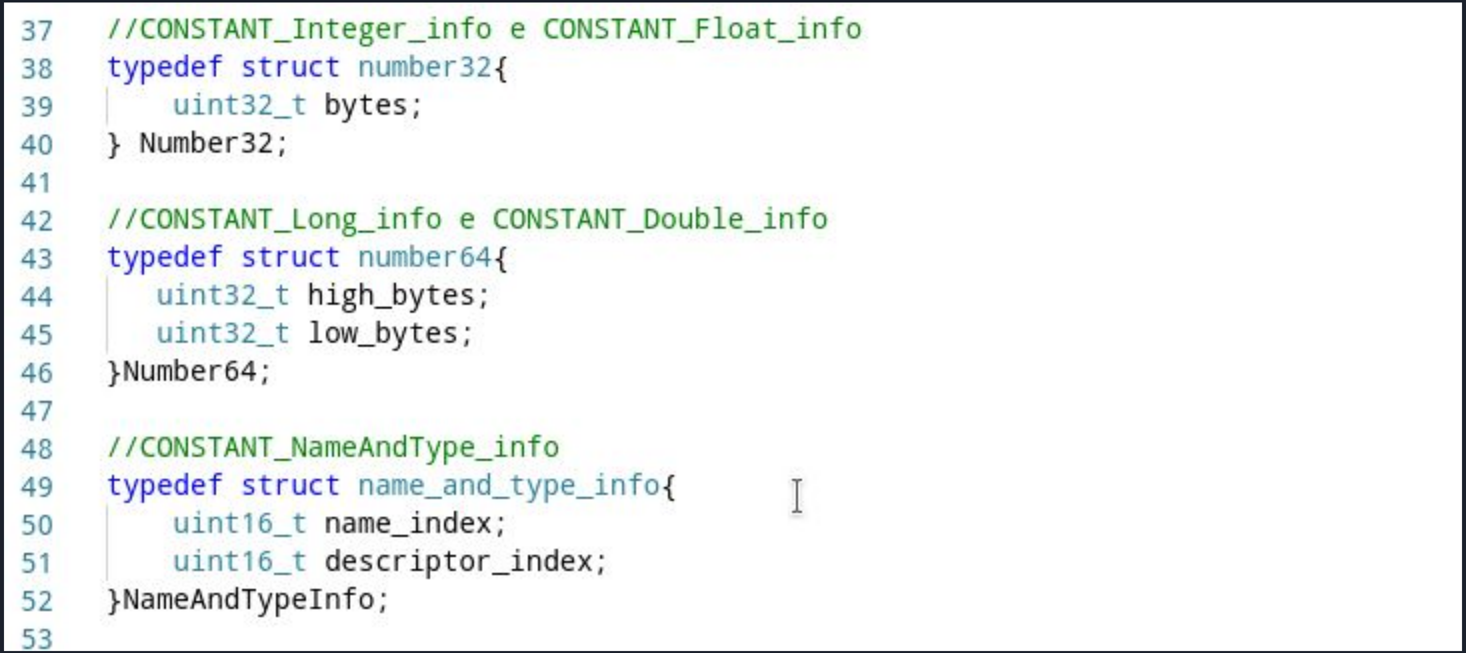


```c

//Constant POOL TAGS
#define CONSTANT_Class 7
#define CONSTANT_Fieldref 9
#define CONSTANT_Methodref 10
#define CONSTANT_InterfaceMethodref 11
#define CONSTANT_String 8
#define CONSTANT_Integer 3
#define CONSTANT_Float 4
#define CONSTANT_Long 5
#define CONSTANT_Double 6
#define CONSTANT_NameAndType 12
#define CONSTANT_Utf8 1
#define CONSTANT_MethodHandle 15
#define CONSTANT_MethodType 16
#define CONSTANT_InvokeDynamic 18

```



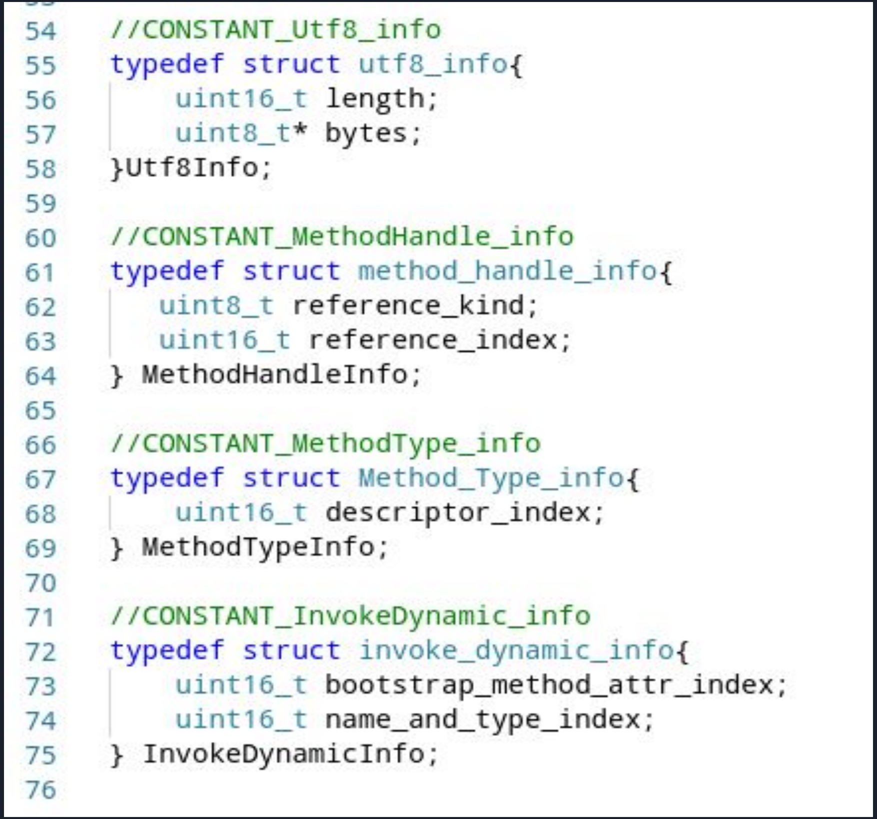
```c
//para CONSTANT_Class_info
typedef struct class_info{
    uint16_t name_index;
} ClassInfo;

//referencias para Fiedref_info, Methodref_info e InterfaceMethodref_info
typedef struct references{
    uint16_t class_index;
    uint16_t name_and_type_index;
}Ref;

//CONSTANT_String_info
typedef struct string_info{
    uint16_t string_index;
} StringInfo;

```

j8_constant_pool.h



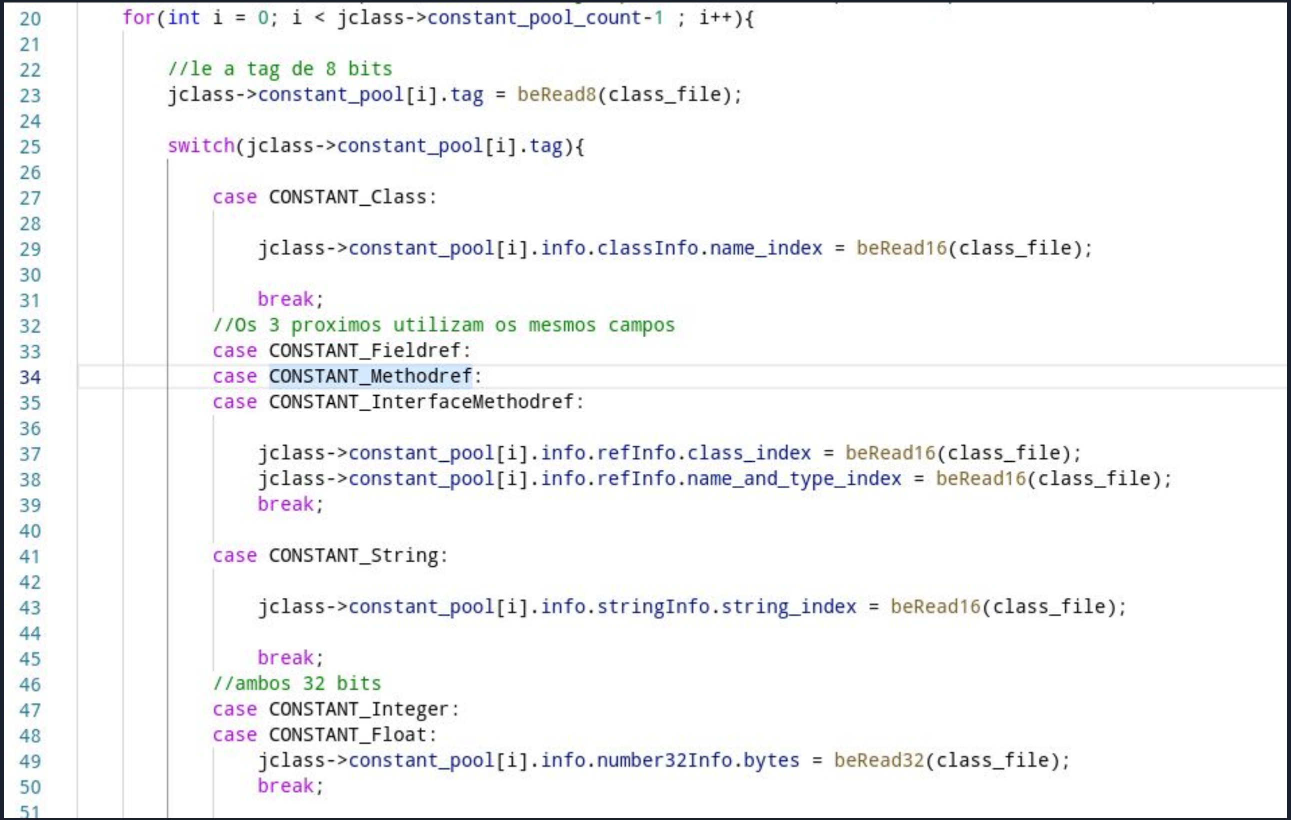
```c
//CONSTANT_Integer_info e CONSTANT_Float_info
typedef struct number32{
    uint32_t bytes;
} Number32;

//CONSTANT_Long_info e CONSTANT_Double_info
typedef struct number64{
   uint32_t high_bytes;
   uint32_t low_bytes;
}Number64;

//CONSTANT_NameAndType_info
typedef struct name_and_type_info{
    uint16_t name_index;
    uint16_t descriptor_index;
}NameAndTypeInfo;

```

j8_constant_pool.h



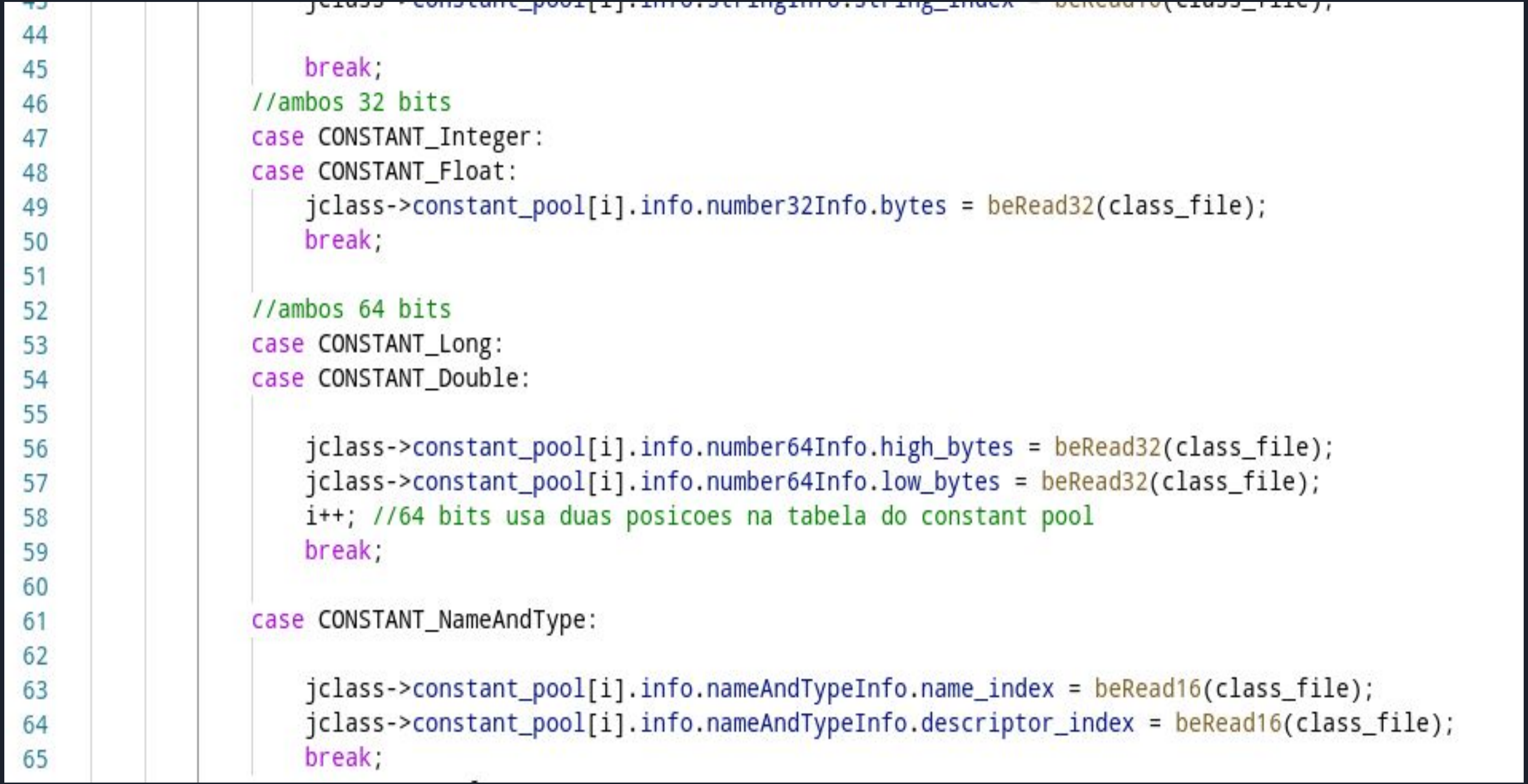
```
//CONSTANT_Utf8_info
typedef struct utf8_info{
    uint16_t length;
    uint8_t* bytes;
}Utf8Info;

//CONSTANT_MethodHandle_info
typedef struct method_handle_info{
   uint8_t reference_kind;
   uint16_t reference_index;
} MethodHandleInfo;

//CONSTANT_MethodType_info
typedef struct Method_Type_info{
    uint16_t descriptor_index;
} MethodTypeInfo;

//CONSTANT_InvokeDynamic_info
typedef struct invoke_dynamic_info{
    uint16_t bootstrap_method_attr_index;
    uint16_t name_and_type_index;
} InvokeDynamicInfo;
```

Como ler o constant pool



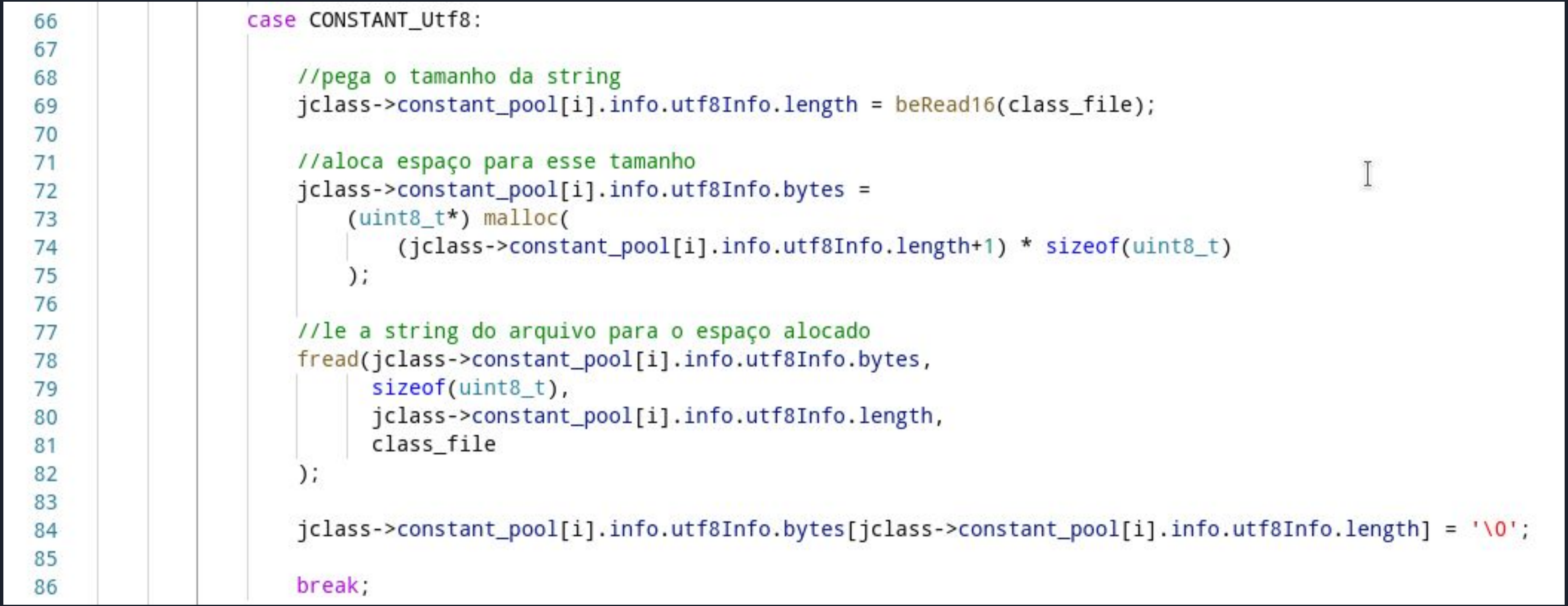
```
for(int i = 0; i < jclass->constant_pool_count-1 ; i++){

    //le a tag de 8 bits
    jclass->constant_pool[i].tag = beRead8(class_file);

    switch(jclass->constant_pool[i].tag){

        case CONSTANT_Class:

            jclass->constant_pool[i].info.classInfo.name_index = beRead16(class_file);

            break;
        //Os 3 proximos utilizam os mesmos campos
        case CONSTANT_Fieldref:
        case CONSTANT_Methodref:
        case CONSTANT_InterfaceMethodref:

            jclass->constant_pool[i].info.refInfo.class_index = beRead16(class_file);
            jclass->constant_pool[i].info.refInfo.name_and_type_index = beRead16(class_file);
            break;

        case CONSTANT_String:

            jclass->constant_pool[i].info.stringInfo.string_index = beRead16(class_file);

            break;
        //ambos 32 bits
        case CONSTANT_Integer:
        case CONSTANT_Float:
            jclass->constant_pool[i].info.number32Info.bytes = beRead32(class_file);
            break;
```

Como ler o constant pool



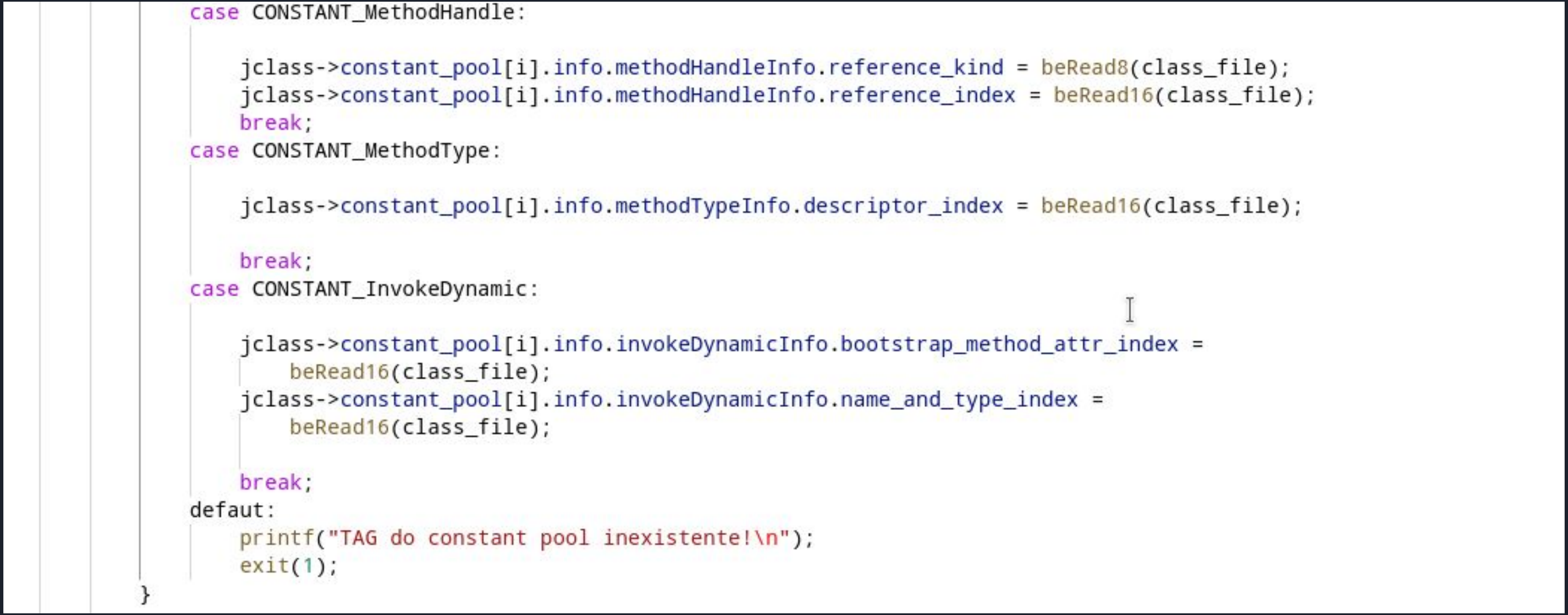
```
        break;
    //ambos 32 bits
    case CONSTANT_Integer:
    case CONSTANT_Float:
        jclass->constant_pool[i].info.number32Info.bytes = beRead32(class_file);
        break;

    //ambos 64 bits
    case CONSTANT_Long:
    case CONSTANT_Double:

        jclass->constant_pool[i].info.number64Info.high_bytes = beRead32(class_file);
        jclass->constant_pool[i].info.number64Info.low_bytes = beRead32(class_file);
        i++; //64 bits usa duas posicoes na tabela do constant pool
        break;

    case CONSTANT_NameAndType:

        jclass->constant_pool[i].info.nameAndTypeInfo.name_index = beRead16(class_file);
        jclass->constant_pool[i].info.nameAndTypeInfo.descriptor_index = beRead16(class_file);
        break;
```

Como ler o constant pool



```c
        case CONSTANT_Utf8:

            //pega o tamanho da string
            jclass->constant_pool[i].info.utf8Info.length = beRead16(class_file);

            //aloca espaço para esse tamanho
            jclass->constant_pool[i].info.utf8Info.bytes =
                (uint8_t*) malloc(
                    (jclass->constant_pool[i].info.utf8Info.length+1) * sizeof(uint8_t)
                );

            //le a string do arquivo para o espaço alocado
            fread(jclass->constant_pool[i].info.utf8Info.bytes,
                  sizeof(uint8_t),
                  jclass->constant_pool[i].info.utf8Info.length,
                  class_file
            );

            jclass->constant_pool[i].info.utf8Info.bytes[jclass->constant_pool[i].info.utf8Info.length] = '\0';

            break;
```

Como ler o constant pool



```
        case CONSTANT_MethodHandle:

            jclass->constant_pool[i].info.methodHandleInfo.reference_kind = beRead8(class_file);
            jclass->constant_pool[i].info.methodHandleInfo.reference_index = beRead16(class_file);
            break;
        case CONSTANT_MethodType:

            jclass->constant_pool[i].info.methodTypeInfo.descriptor_index = beRead16(class_file);

            break;
        case CONSTANT_InvokeDynamic:

            jclass->constant_pool[i].info.invokeDynamicInfo.bootstrap_method_attr_index =
                beRead16(class_file);
            jclass->constant_pool[i].info.invokeDynamicInfo.name_and_type_index =
                beRead16(class_file);

            break;
        defaut:
            printf("TAG do constant pool inexistente!\n");
            exit(1);
    }
```

Como fica o Constant Pool na saída

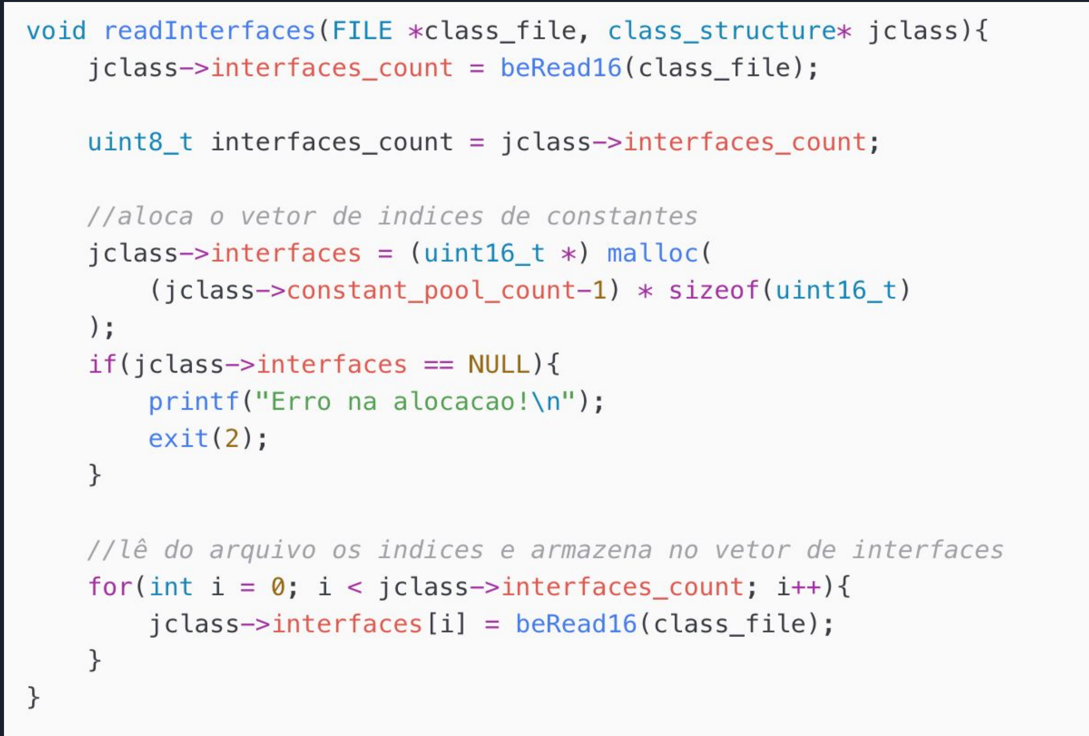
```
Constant Pool Count: 31
Mostrando o conteúdo do constant pool:
        #1: Methodref   class index: #9, name and type index: #18
        #2: Double      high bytes: 0x40588f5c, low bytes: 0x28f5c28f
        #4: Double      high bytes: 0x4060dbd7, low bytes: 0xa3d70a4
        #6: Fieldref      class index: #19, name and type index: #20
        #7: Methodref   class index: #21, name and type index: #22
        #8: Class         name Index: #23
        #9: Class         name Index: #24
        #10: Utf8         length: #6, bytes: "<init>"
        #11: Utf8         length: #3, bytes: "()V"
        #12: Utf8         length: #4, bytes: "Code"
        #13: Utf8         length: #15, bytes: "LineNumberTable"
        #14: Utf8         length: #4, bytes: "main"
        #15: Utf8         length: #22, bytes: "([Ljava/lang/String;)V"
        #16: Utf8         length: #10, bytes: "SourceFile"
```

Como fica o Constant Pool na saída

```
#17: Utf8             length: #22, bytes: "double_aritmetica.java"
#18: NameAndType      name index: #10, descriptor_index: #11
#19: Class            name Index: #25
#20: NameAndType      name index: #26, descriptor_index: #27
#21: Class            name Index: #28
#22: NameAndType      name index: #29, descriptor_index: #30
#23: Utf8             length: #17, bytes: "double_aritmetica"
#24: Utf8             length: #16, bytes: "java/lang/Object"
#25: Utf8             length: #16, bytes: "java/lang/System"
#26: Utf8             length: #3, bytes: "out"
#27: Utf8             length: #21, bytes: "Ljava/io/PrintStream;"
#28: Utf8             length: #19, bytes: "java/io/PrintStream"
#29: Utf8             length: #7, bytes: "println"
#30: Utf8             length: #4, bytes: "(D)V"
```

Interfaces

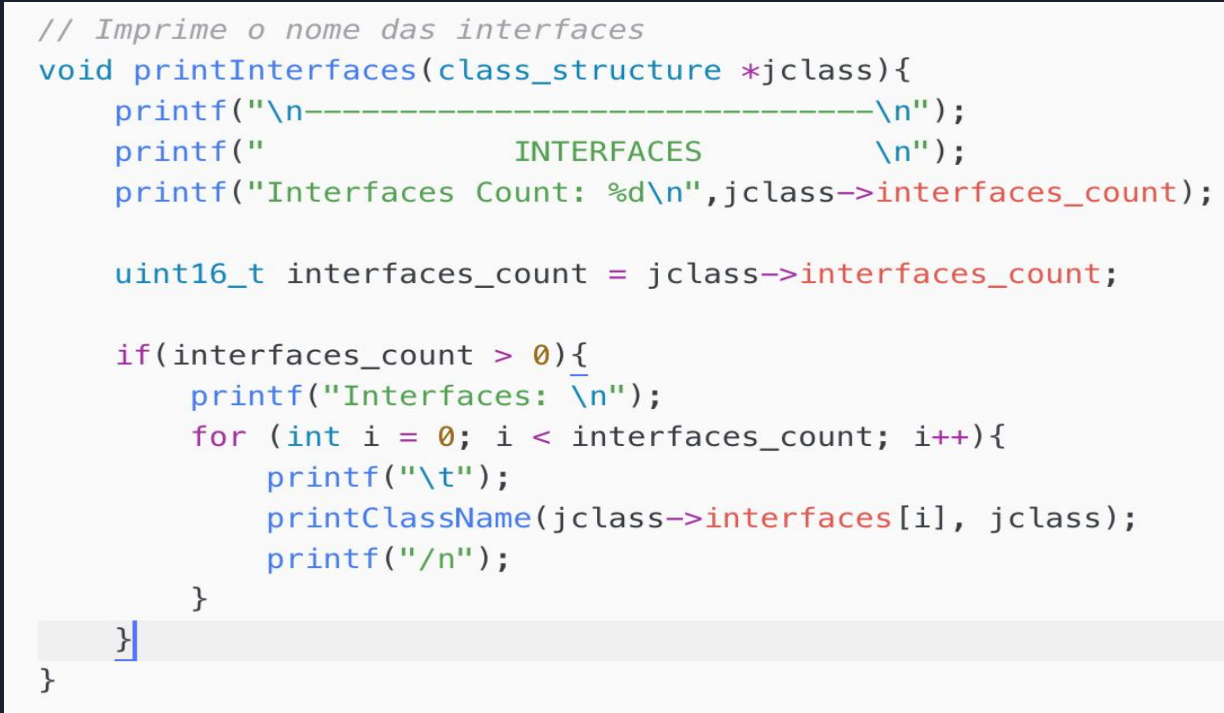
- readInterfaces
    - Realiza a leitura do número de interfaces
    - Alocação de memória para o número de interfaces



```
void readInterfaces(FILE *class_file, class_structure* jclass){
    jclass->interfaces_count = beRead16(class_file);

    uint8_t interfaces_count = jclass->interfaces_count;

    //aloca o vetor de indices de constantes
    jclass->interfaces = (uint16_t *) malloc(
        (jclass->constant_pool_count-1) * sizeof(uint16_t)
    );
    if(jclass->interfaces == NULL){
        printf("Erro na alocacao!\n");
        exit(2);
    }

    //lê do arquivo os indices e armazena no vetor de interfaces
    for(int i = 0; i < jclass->interfaces_count; i++){
        jclass->interfaces[i] = beRead16(class_file);
    }
}
```

Interfaces

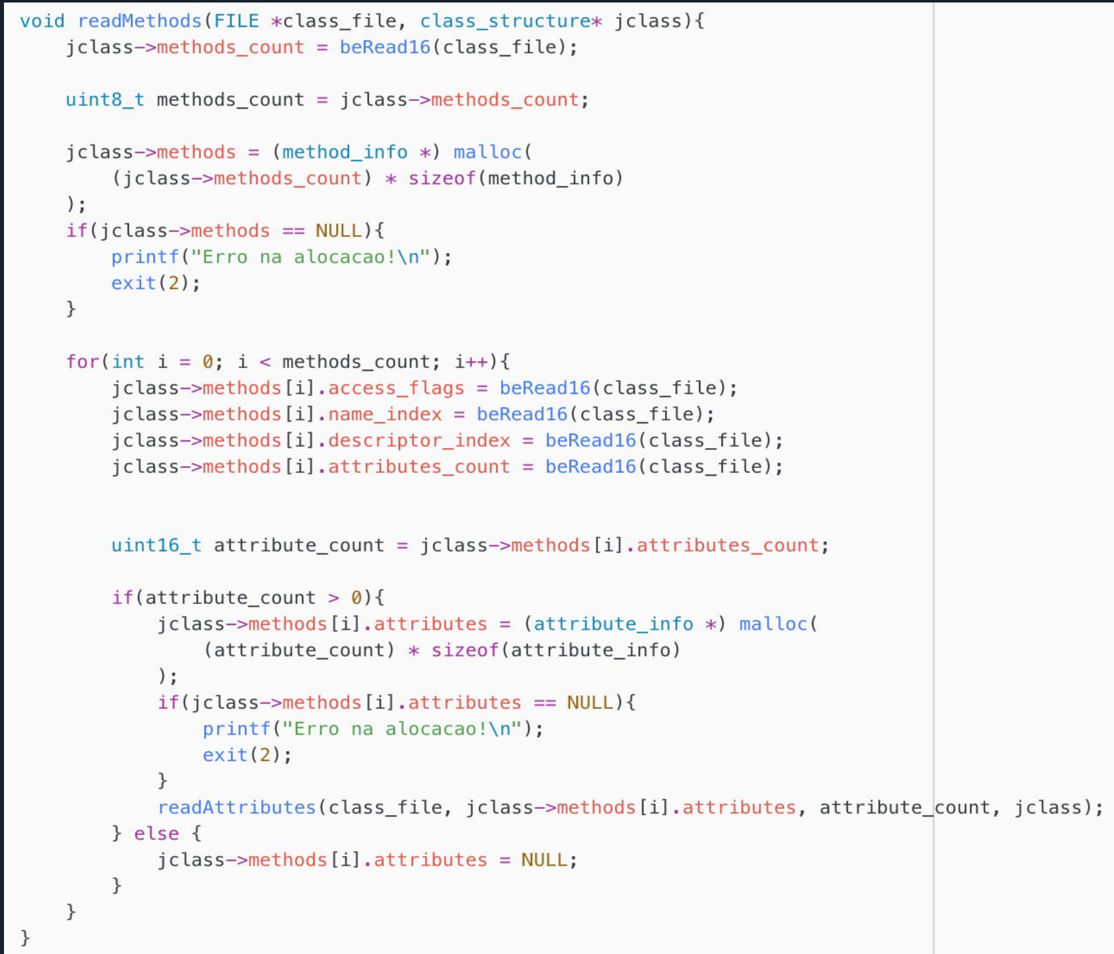
- printInterfaces
  - Realiza a impressão de todas as interfaces



```c
// Imprime o nome das interfaces
void printInterfaces(class_structure *jclass){
    printf("\n------------------------------\n");
    printf("                 INTERFACES          \n");
    printf("Interfaces Count: %d\n",jclass->interfaces_count);

    uint16_t interfaces_count = jclass->interfaces_count;

    if(interfaces_count > 0){
        printf("Interfaces: \n");
        for (int i = 0; i < interfaces_count; i++){
            printf("\t");
            printClassName(jclass->interfaces[i], jclass);
            printf("/n");
        }
    }
}
```

Methods

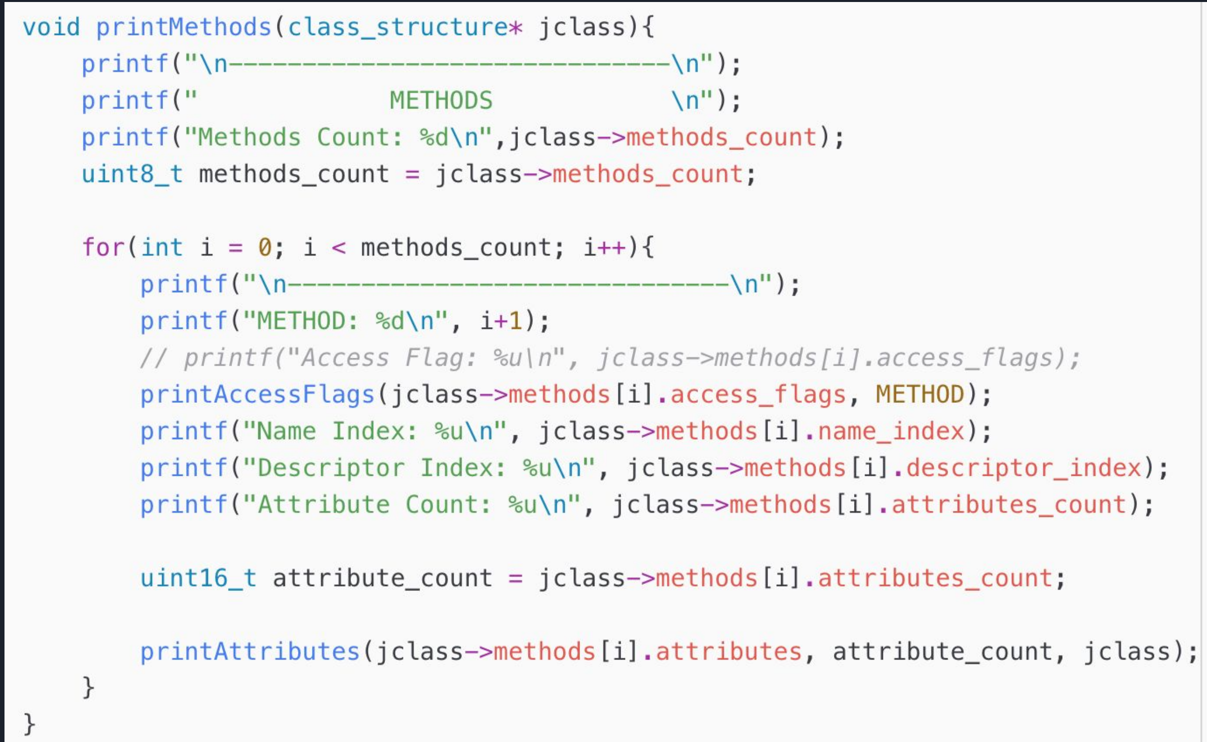
- readMethods
  - Realiza a leitura do número de methods
  - Alocação de memória:
    - Número de methods
    - Número de atributtes



```c
void readMethods(FILE *class_file, class_structure* jclass){
    jclass->methods_count = beRead16(class_file);

    uint8_t methods_count = jclass->methods_count;

    jclass->methods = (method_info *) malloc(
        (jclass->methods_count) * sizeof(method_info)
    );
    if(jclass->methods == NULL){
        printf("Erro na alocacao!\n");
        exit(2);
    }

    for(int i = 0; i < methods_count; i++){
        jclass->methods[i].access_flags = beRead16(class_file);
        jclass->methods[i].name_index = beRead16(class_file);
        jclass->methods[i].descriptor_index = beRead16(class_file);
        jclass->methods[i].attributes_count = beRead16(class_file);


        uint16_t attribute_count = jclass->methods[i].attributes_count;

        if(attribute_count > 0){
            jclass->methods[i].attributes = (attribute_info *) malloc(
                (attribute_count) * sizeof(attribute_info)
            );
            if(jclass->methods[i].attributes == NULL){
                printf("Erro na alocacao!\n");
                exit(2);
            }
            readAttributes(class_file, jclass->methods[i].attributes, attribute_count, jclass);
        } else {
            jclass->methods[i].attributes = NULL;
        }
    }
}
```

Methods

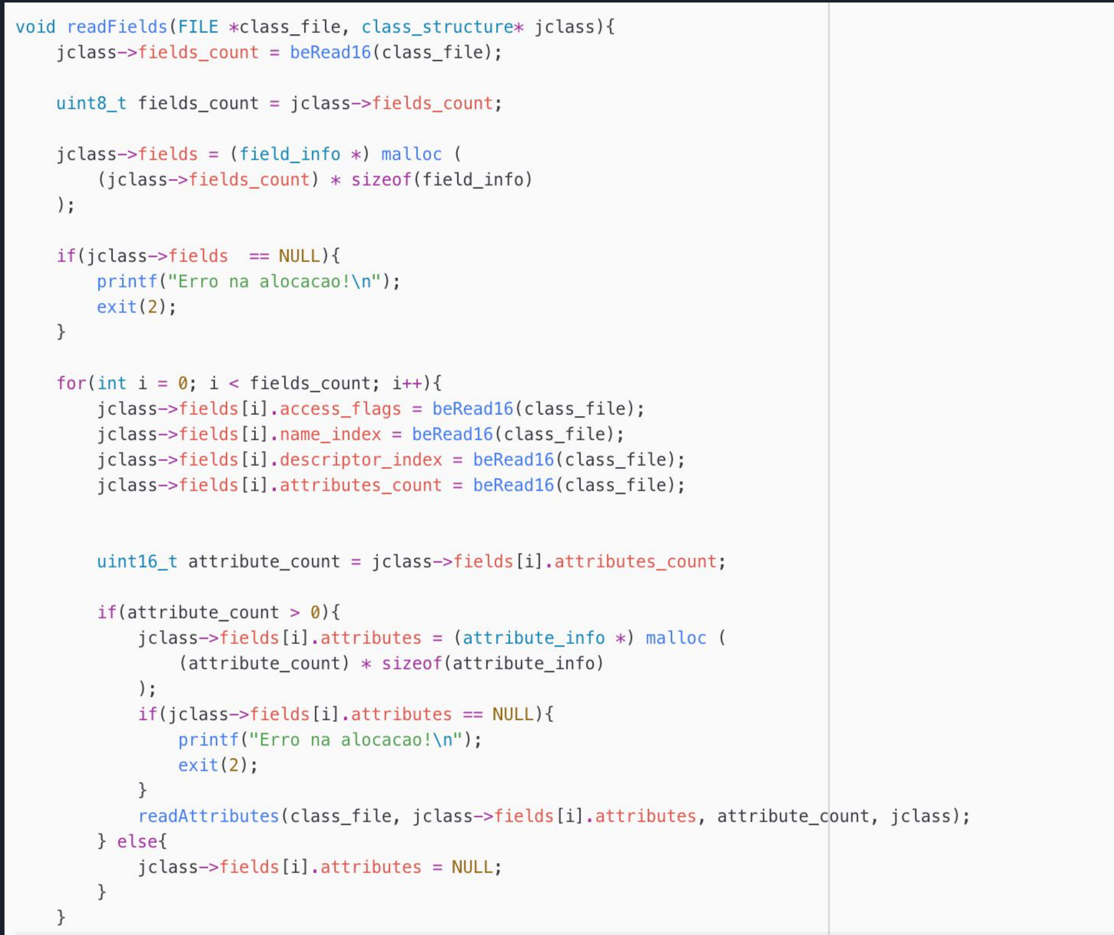
- printMethods
  - Realiza a impressão de todos os metodos



```
void printMethods(class_structure* jclass){
    printf("\n------------------------------\n");
    printf("             METHODS            \n");
    printf("Methods Count: %d\n",jclass->methods_count);
    uint8_t methods_count = jclass->methods_count;

    for(int i = 0; i < methods_count; i++){
        printf("\n------------------------------\n");
        printf("METHOD: %d\n", i+1);
        // printf("Access Flag: %u\n", jclass->methods[i].access_flags);
        printAccessFlags(jclass->methods[i].access_flags, METHOD);
        printf("Name Index: %u\n", jclass->methods[i].name_index);
        printf("Descriptor Index: %u\n", jclass->methods[i].descriptor_index);
        printf("Attribute Count: %u\n", jclass->methods[i].attributes_count);

        uint16_t attribute_count = jclass->methods[i].attributes_count;

        printAttributes(jclass->methods[i].attributes, attribute_count, jclass);
    }
}
```

Fields

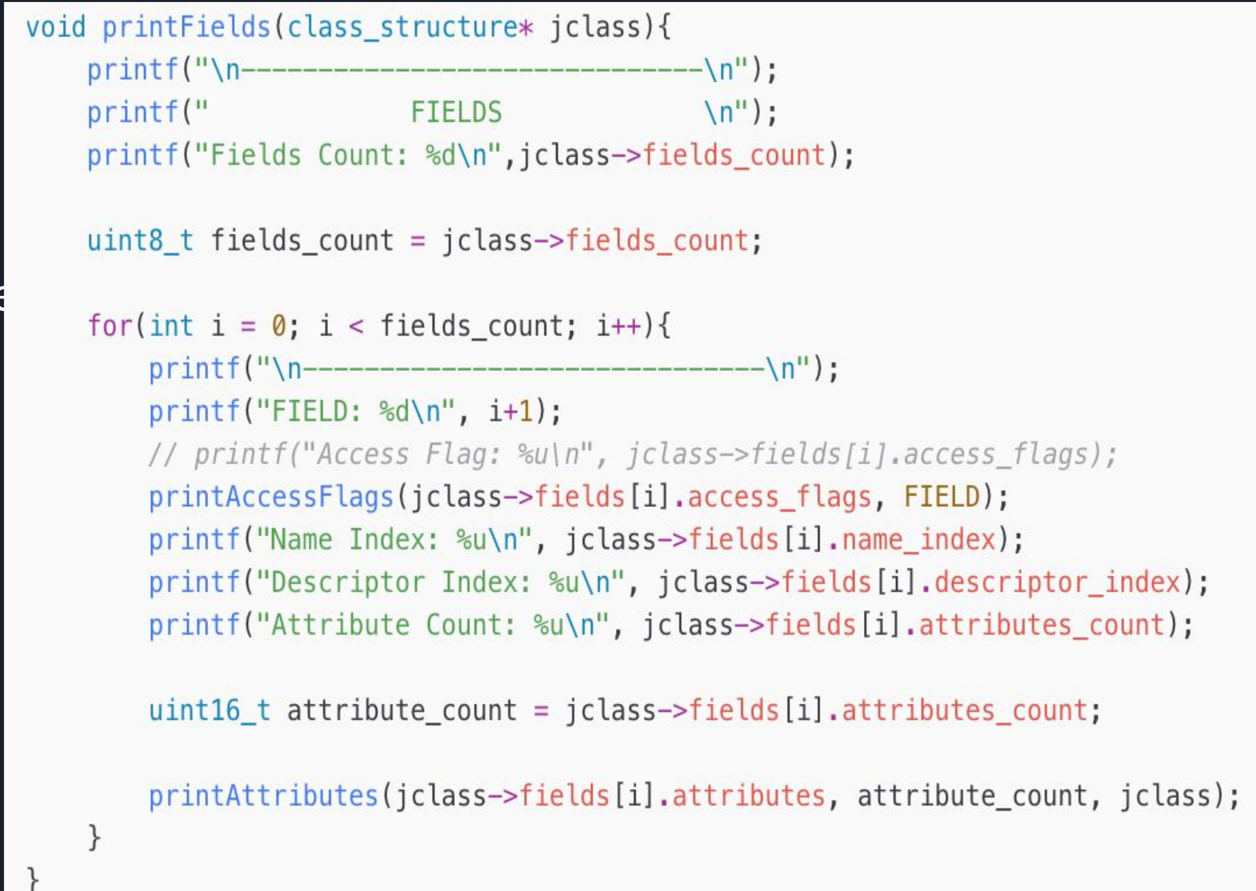
- readFields
  - Realiza a leitura do número de fields
  - Alocação de memória:
    - Número de fields
    - Número de atributtes



```c
void readFields(FILE *class_file, class_structure* jclass){
    jclass->fields_count = beRead16(class_file);

    uint8_t fields_count = jclass->fields_count;

    jclass->fields = (field_info *) malloc (
        (jclass->fields_count) * sizeof(field_info)
    );

    if(jclass->fields  == NULL){
        printf("Erro na alocacao!\n");
        exit(2);
    }

    for(int i = 0; i < fields_count; i++){
        jclass->fields[i].access_flags = beRead16(class_file);
        jclass->fields[i].name_index = beRead16(class_file);
        jclass->fields[i].descriptor_index = beRead16(class_file);
        jclass->fields[i].attributes_count = beRead16(class_file);


        uint16_t attribute_count = jclass->fields[i].attributes_count;

        if(attribute_count > 0){
            jclass->fields[i].attributes = (attribute_info *) malloc (
                (attribute_count) * sizeof(attribute_info)
            );
            if(jclass->fields[i].attributes == NULL){
                printf("Erro na alocacao!\n");
                exit(2);
            }
            readAttributes(class_file, jclass->fields[i].attributes, attribute_count, jclass);
        } else{
            jclass->fields[i].attributes = NULL;
        }
    }
```

Fields

- printFields
  - Realiza a impressão de
    todos os fields

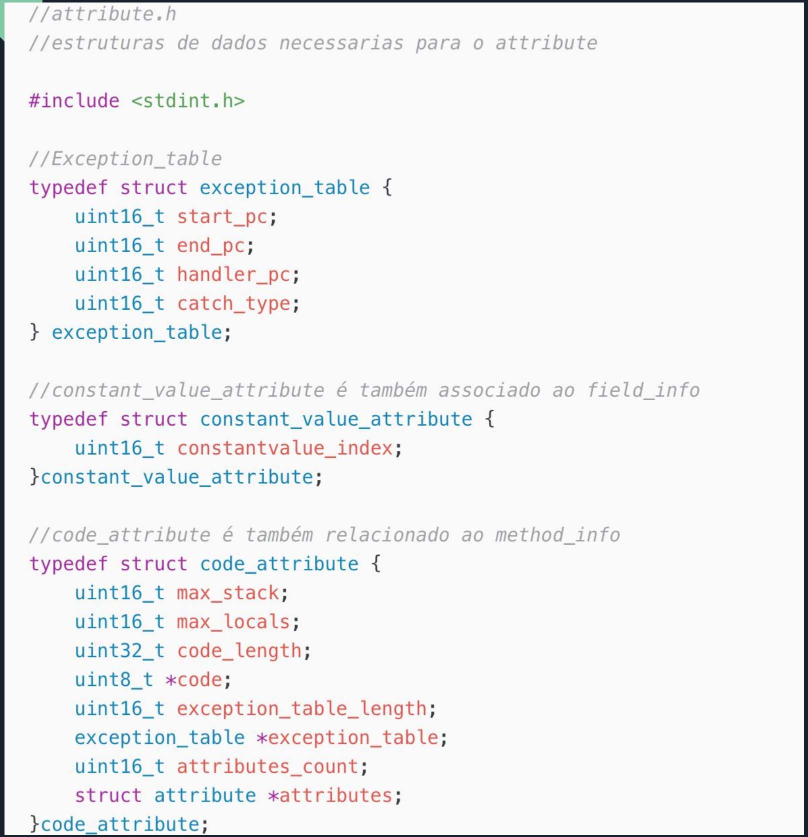


```
void printFields(class_structure* jclass){
    printf("\n-----------------------------\n");
    printf("                FIELDS               \n");
    printf("Fields Count: %d\n",jclass->fields_count);

    uint8_t fields_count = jclass->fields_count;

    for(int i = 0; i < fields_count; i++){
        printf("\n-----------------------------\n");
        printf("FIELD: %d\n", i+1);
        // printf("Access Flag: %u\n", jclass->fields[i].access_flags);
        printAccessFlags(jclass->fields[i].access_flags, FIELD);
        printf("Name Index: %u\n", jclass->fields[i].name_index);
        printf("Descriptor Index: %u\n", jclass->fields[i].descriptor_index);
        printf("Attribute Count: %u\n", jclass->fields[i].attributes_count);

        uint16_t attribute_count = jclass->fields[i].attributes_count;

        printAttributes(jclass->fields[i].attributes, attribute_count, jclass);
    }
}
```

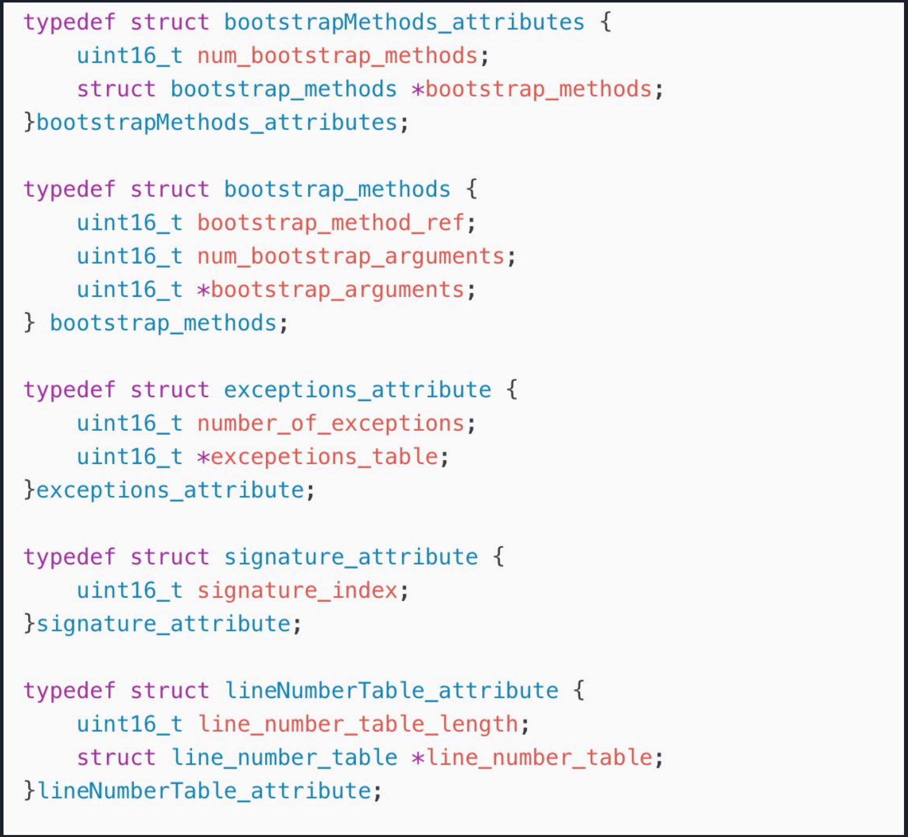
Attributes

- Struct -> Attribute_info
- Union -> info
  - code_attribute
  - constant_value_attribute
  - exceptions_attribute
  - bootstrapMethods_attributes
  - signature_attribute
  - lineNumberTable_attribute
  - sourceFile_attribute
  - innerClasses_attribute
  - localVariableTable_attribute

attributes.h

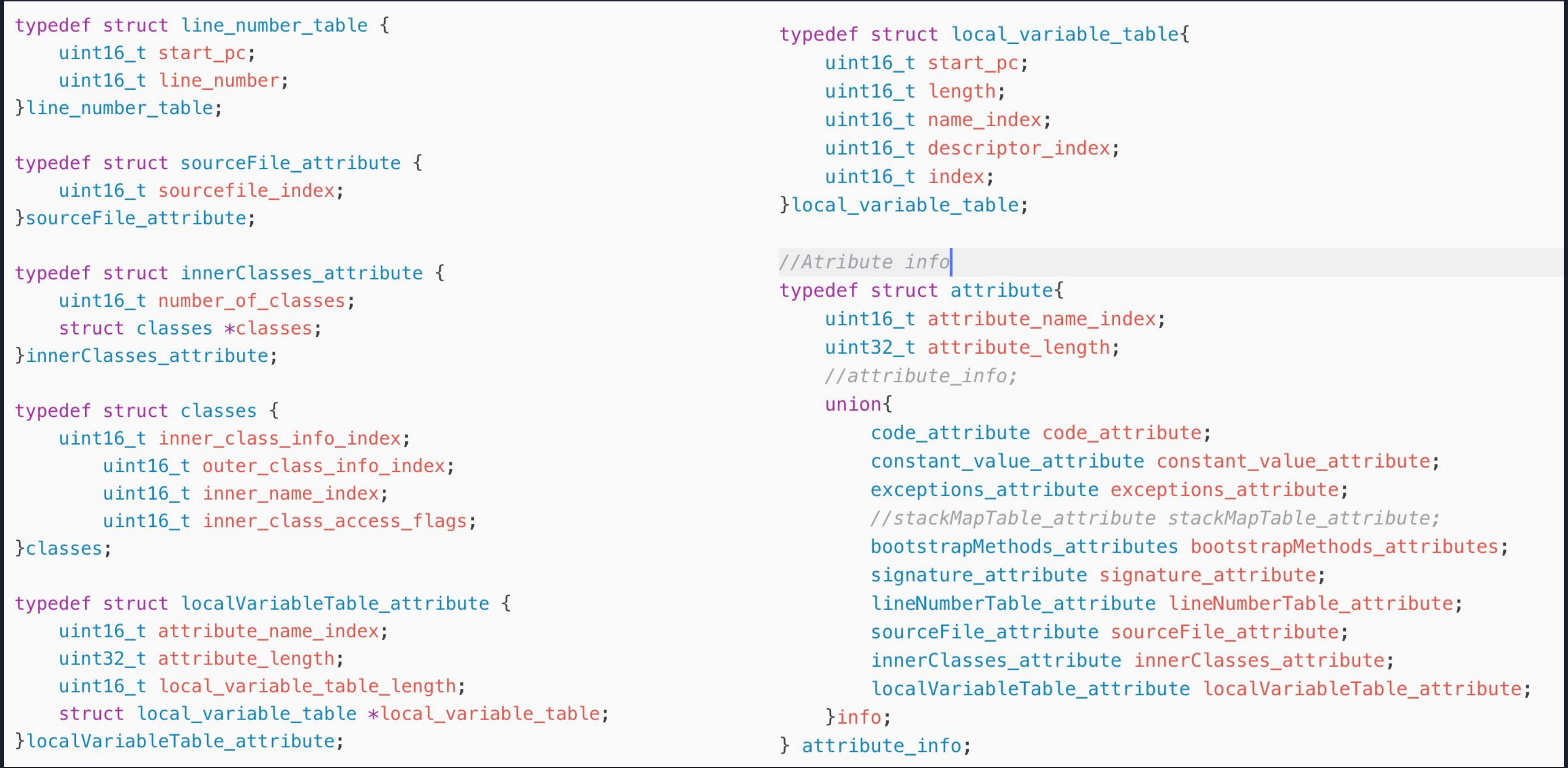


```c
//attribute.h
//estruturas de dados necessarias para o attribute

#include <stdint.h>

//Exception_table
typedef struct exception_table {
    uint16_t start_pc;
    uint16_t end_pc;
    uint16_t handler_pc;
    uint16_t catch_type;
} exception_table;

//constant_value_attribute é também associado ao field_info
typedef struct constant_value_attribute {
    uint16_t constantvalue_index;
}constant_value_attribute;

//code_attribute é também relacionado ao method_info
typedef struct code_attribute {
    uint16_t max_stack;
    uint16_t max_locals;
    uint32_t code_length;
    uint8_t *code;
    uint16_t exception_table_length;
    exception_table *exception_table;
    uint16_t attributes_count;
    struct attribute *attributes;
}code_attribute;
```



```c
typedef struct bootstrapMethods_attributes {
    uint16_t num_bootstrap_methods;
    struct bootstrap_methods *bootstrap_methods;
}bootstrapMethods_attributes;

typedef struct bootstrap_methods {
    uint16_t bootstrap_method_ref;
    uint16_t num_bootstrap_arguments;
    uint16_t *bootstrap_arguments;
} bootstrap_methods;

typedef struct exceptions_attribute {
    uint16_t number_of_exceptions;
    uint16_t *excepetions_table;
}exceptions_attribute;

typedef struct signature_attribute {
    uint16_t signature_index;
}signature_attribute;

typedef struct lineNumberTable_attribute {
    uint16_t line_number_table_length;
    struct line_number_table *line_number_table;
}lineNumberTable_attribute;
```

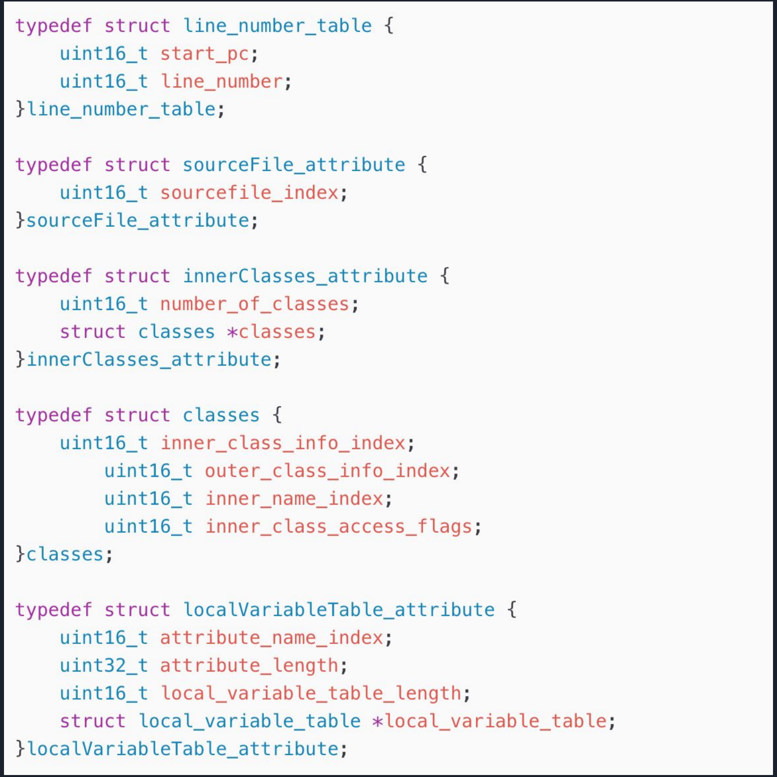
attributes.h



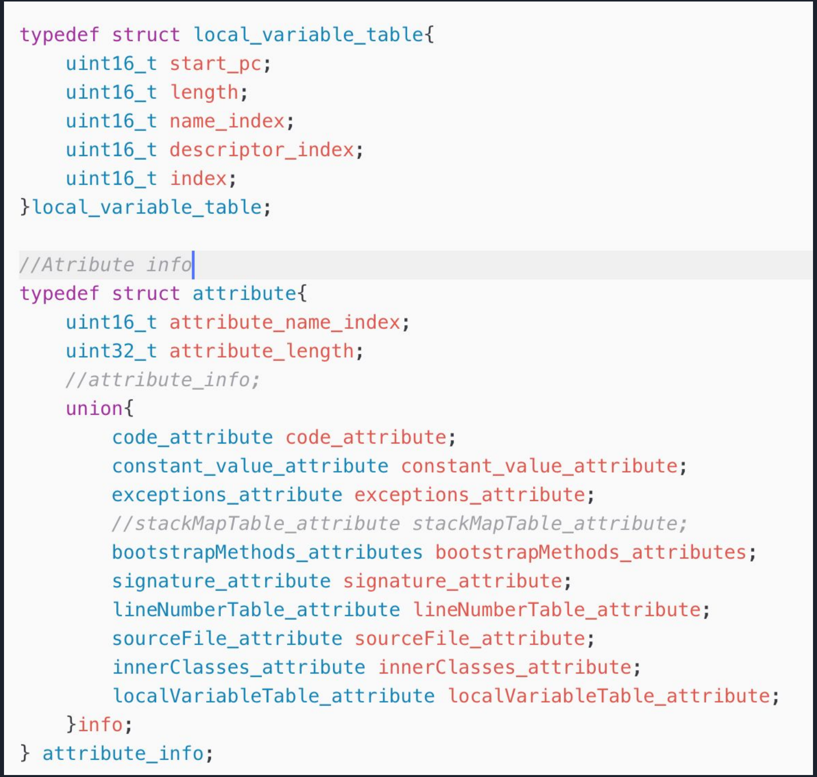
```
typedef struct line_number_table {
    uint16_t start_pc;
    uint16_t line_number;
}line_number_table;

typedef struct sourceFile_attribute {
    uint16_t sourcefile_index;
}sourceFile_attribute;

typedef struct innerClasses_attribute {
    uint16_t number_of_classes;
    struct classes *classes;
}innerClasses_attribute;

typedef struct classes {
    uint16_t inner_class_info_index;
        uint16_t outer_class_info_index;
        uint16_t inner_name_index;
        uint16_t inner_class_access_flags;
}classes;

typedef struct localVariableTable_attribute {
    uint16_t attribute_name_index;
    uint32_t attribute_length;
    uint16_t local_variable_table_length;
    struct local_variable_table *local_variable_table;
}localVariableTable_attribute;
```

```
typedef struct local_variable_table{
    uint16_t start_pc;
    uint16_t length;
    uint16_t name_index;
    uint16_t descriptor_index;
    uint16_t index;
}local_variable_table;

//Atribute info
typedef struct attribute{
    uint16_t attribute_name_index;
    uint32_t attribute_length;
    //attribute_info;
    union{
        code_attribute code_attribute;
        constant_value_attribute constant_value_attribute;
        exceptions_attribute exceptions_attribute;
        //stackMapTable_attribute stackMapTable_attribute;
        bootstrapMethods_attributes bootstrapMethods_attributes;
        signature_attribute signature_attribute;
        lineNumberTable_attribute lineNumberTable_attribute;
        sourceFile_attribute sourceFile_attribute;
        innerClasses_attribute innerClasses_attribute;
        localVariableTable_attribute localVariableTable_attribute;
    }info;
} attribute_info;
```

attributes.h



```c
typedef struct line_number_table {
    uint16_t start_pc;
    uint16_t line_number;
}line_number_table;

typedef struct sourceFile_attribute {
    uint16_t sourcefile_index;
}sourceFile_attribute;

typedef struct innerClasses_attribute {
    uint16_t number_of_classes;
    struct classes *classes;
}innerClasses_attribute;

typedef struct classes {
    uint16_t inner_class_info_index;
        uint16_t outer_class_info_index;
        uint16_t inner_name_index;
        uint16_t inner_class_access_flags;
}classes;

typedef struct localVariableTable_attribute {
    uint16_t attribute_name_index;
    uint32_t attribute_length;
    uint16_t local_variable_table_length;
    struct local_variable_table *local_variable_table;
}localVariableTable_attribute;
```



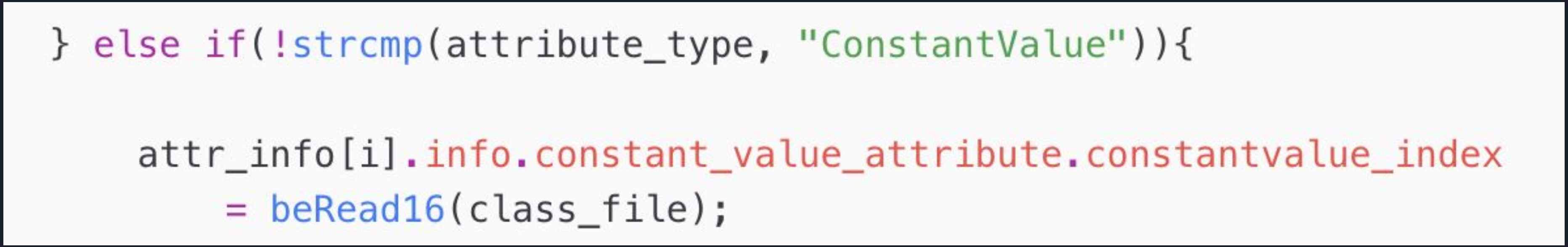
```c
typedef struct local_variable_table{
    uint16_t start_pc;
    uint16_t length;
    uint16_t name_index;
    uint16_t descriptor_index;
    uint16_t index;
}local_variable_table;

//Atribute info
typedef struct attribute{
    uint16_t attribute_name_index;
    uint32_t attribute_length;
    //attribute_info;
    union{
        code_attribute code_attribute;
        constant_value_attribute constant_value_attribute;
        exceptions_attribute exceptions_attribute;
        //stackMapTable_attribute stackMapTable_attribute;
        bootstrapMethods_attributes bootstrapMethods_attributes;
        signature_attribute signature_attribute;
        lineNumberTable_attribute lineNumberTable_attribute;
        sourceFile_attribute sourceFile_attribute;
        innerClasses_attribute innerClasses_attribute;
        localVariableTable_attribute localVariableTable_attribute;
    }info;
} attribute_info;
```

readAttributes

- Função que realiza a leitura dos tipos attributes
- Caso especial de *Code* (leitura recursiva)
- Realiza a leitura de todos os elementos de *union info*
- Utilizado *strcmp* (*string.h*) para comparação dos tipos attributes

readAttributes

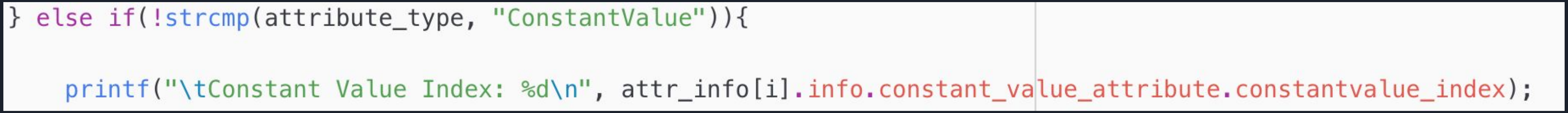
Exemplo:



```
} else if(!strcmp(attribute_type, "ConstantValue")){

    attr_info[i].info.constant_value_attribute.constantvalue_index
        = beRead16(class_file);
```

printAttributes

- Realiza a impressão de todos os attributes
- Utiliza strcmp (string.h)
- Exemplo:



```
} else if(!strcmp(attribute_type, "ConstantValue")){

    printf("\tConstant Value Index: %d\n", attr_info[i].info.constant_value_attribute.constantvalue_index);
```

Integrantes

Paulo Borges Teixeira Neto	15/0143753
Lucas Campos Jorge	15/0154135
João Marcelo Nunes Chaves	15/0132085
Christian Luis Marcondes Costa	15/0153538
Rodrigo Demetrio	15/0147384