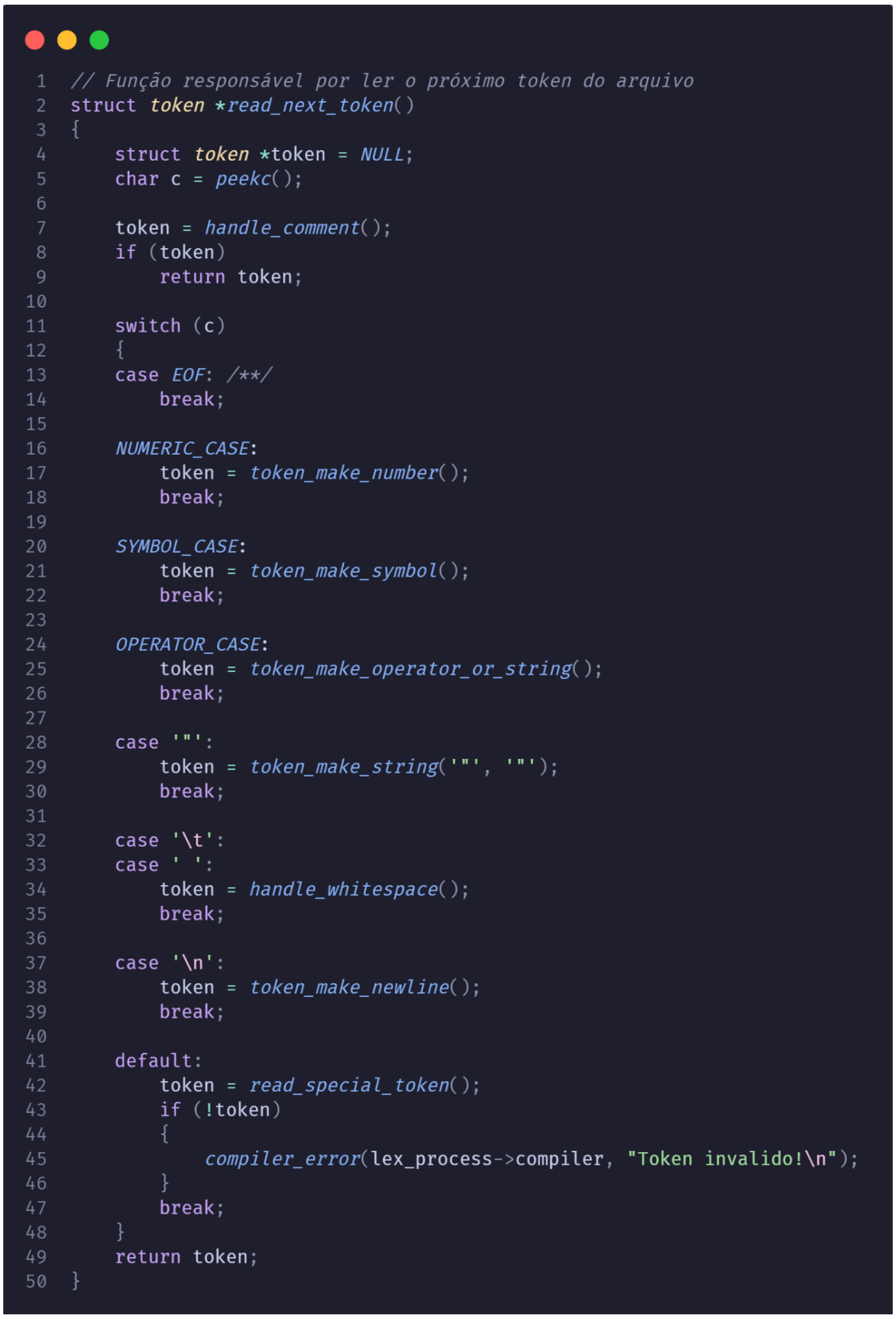
Compiladores - Turma A - 0225 - Noturno
Prof. Thiago Fernandes Neves

# ENTREGA LAB03

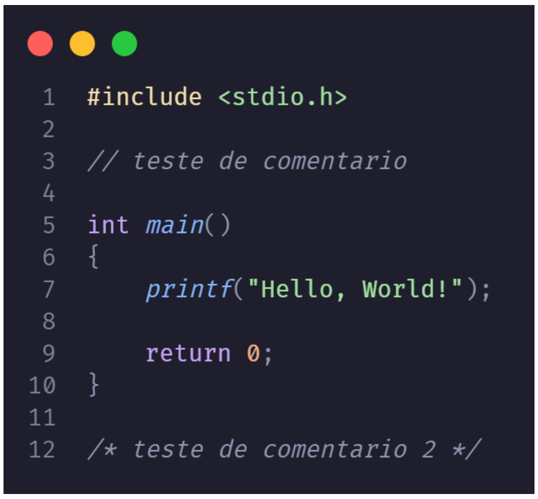
**Integrantes:**
- Isabela Garcia Godinho
- João Victor Azevedo dos Santos
- Yago Péres dos Santos

1. **Implementação de Todos os Tipos de Tokens**



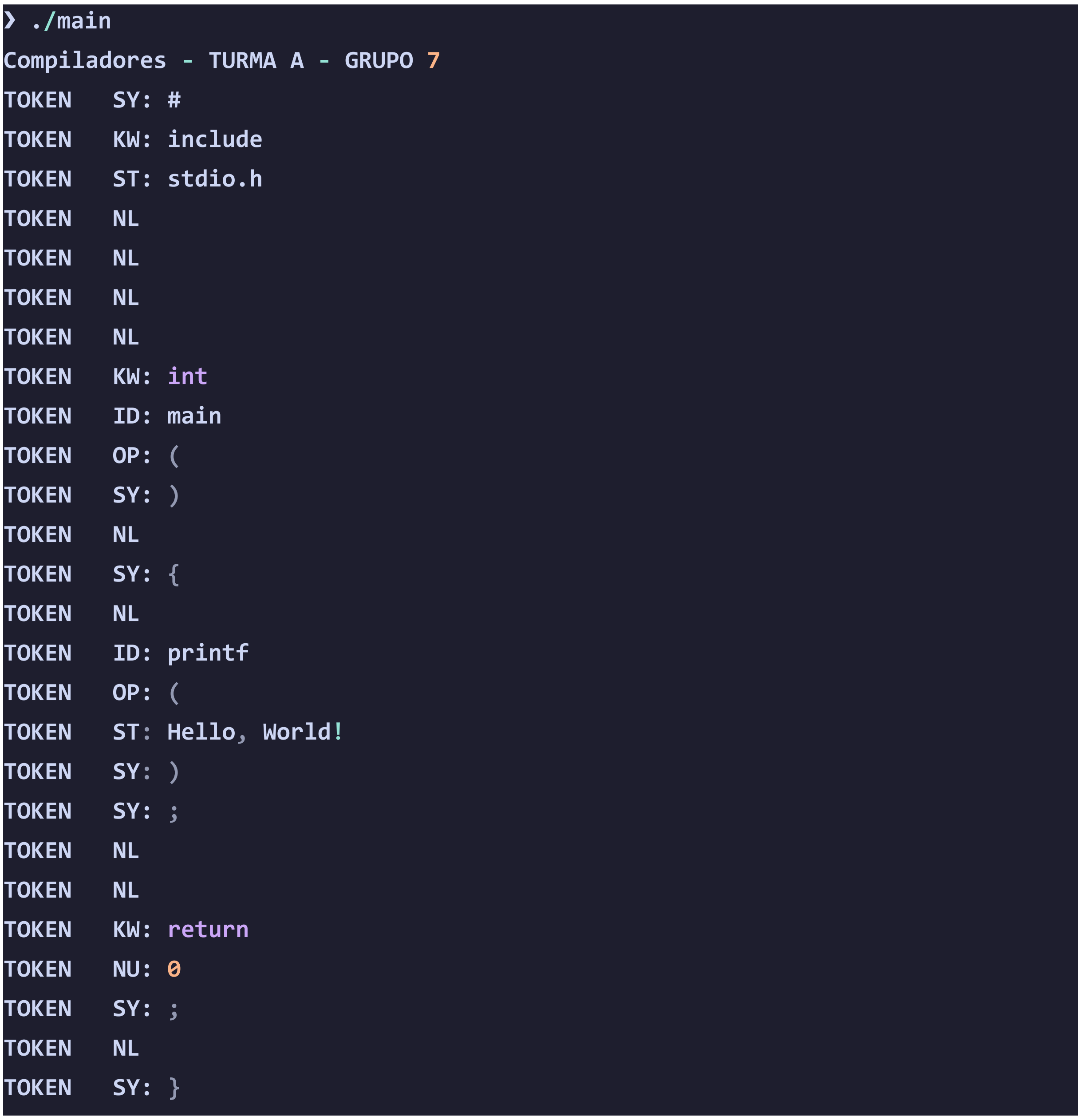
```c
// Função responsável por ler o próximo token do arquivo
struct token *read_next_token()
{
    struct token *token = NULL;
    char c = peekc();

    token = handle_comment();
    if (token)
        return token;

    switch (c)
    {
    case EOF: /**/
        break;

    NUMERIC_CASE:
        token = token_make_number();
        break;

    SYMBOL_CASE:
        token = token_make_symbol();
        break;

    OPERATOR_CASE:
        token = token_make_operator_or_string();
        break;

    case '"':
        token = token_make_string('"', '"');
        break;

    case '\t':
    case ' ':
        token = handle_whitespace();
        break;

    case '\n':
        token = token_make_newline();
        break;

    default:
        token = read_special_token();
        if (!token)
        {
            compiler_error(lex_process->compiler, "Token invalido!\n");
        }
        break;
    }
    return token;
}
```

2. **Realizar os testes**
    a.

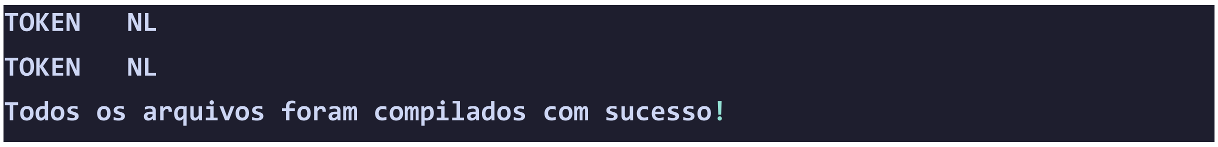


```c
#include <stdio.h>

// teste de comentario

int main()
{
    printf("Hello, World!");

    return 0;
}

/* teste de comentario 2 */
```

**Resultados:**

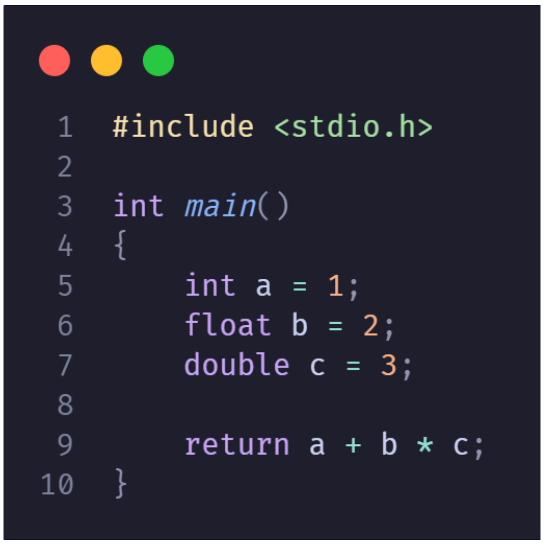


```
❯ ./main
Compiladores - TURMA A - GRUPO 7
TOKEN   SY: #
TOKEN   KW: include
TOKEN   ST: stdio.h
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   KW: int
TOKEN   ID: main
TOKEN   OP: (
TOKEN   SY: )
TOKEN   NL
TOKEN   SY: {
TOKEN   NL
TOKEN   ID: printf
TOKEN   OP: (
TOKEN   ST: Hello, World!
TOKEN   SY: )
TOKEN   SY: ;
TOKEN   NL
TOKEN   NL
TOKEN   KW: return
TOKEN   NU: 0
TOKEN   SY: ;
TOKEN   NL
TOKEN   SY: }
```



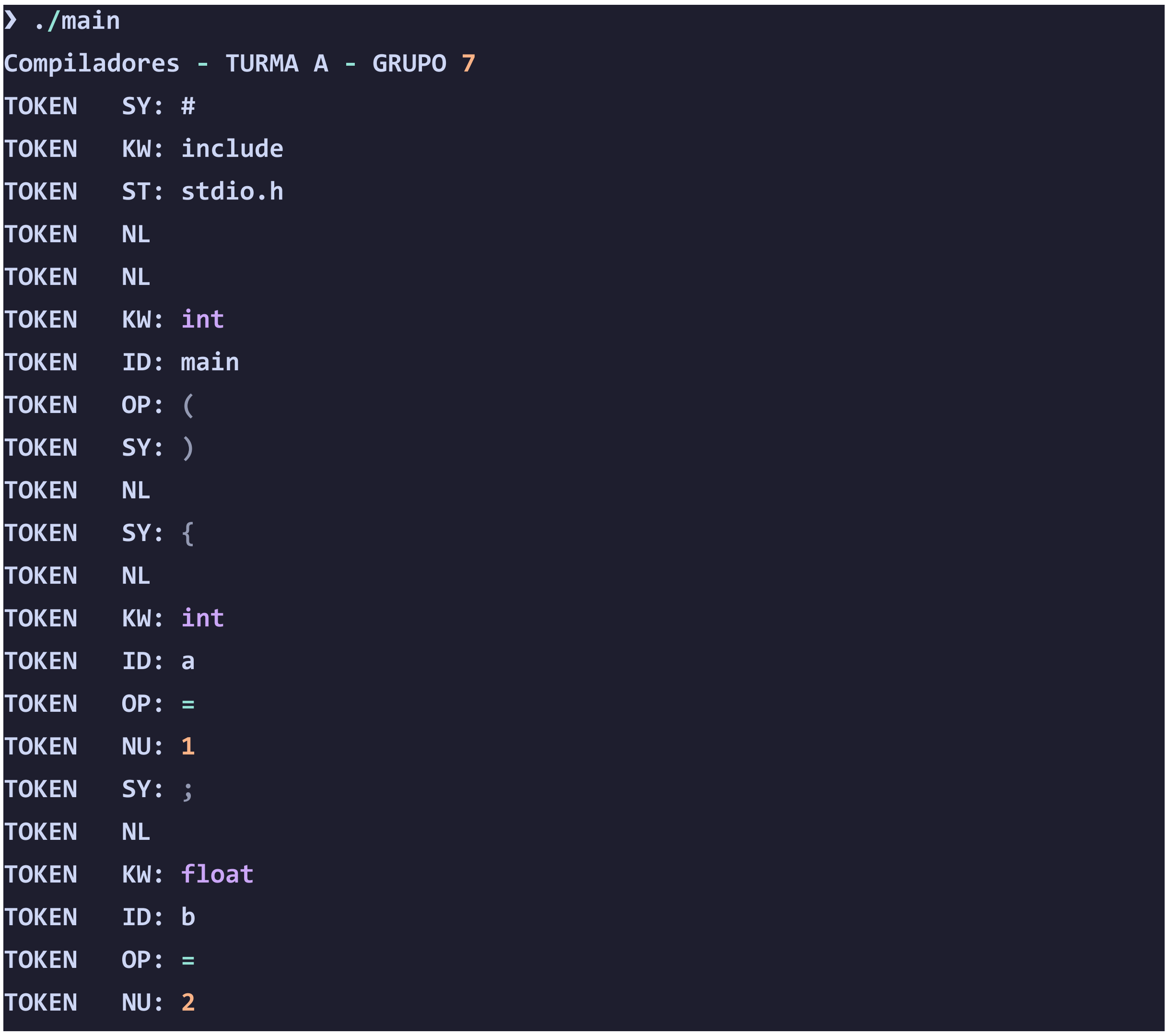
```
TOKEN    NL
TOKEN    NL
Todos os arquivos foram compilados com sucesso!
```

**b.**

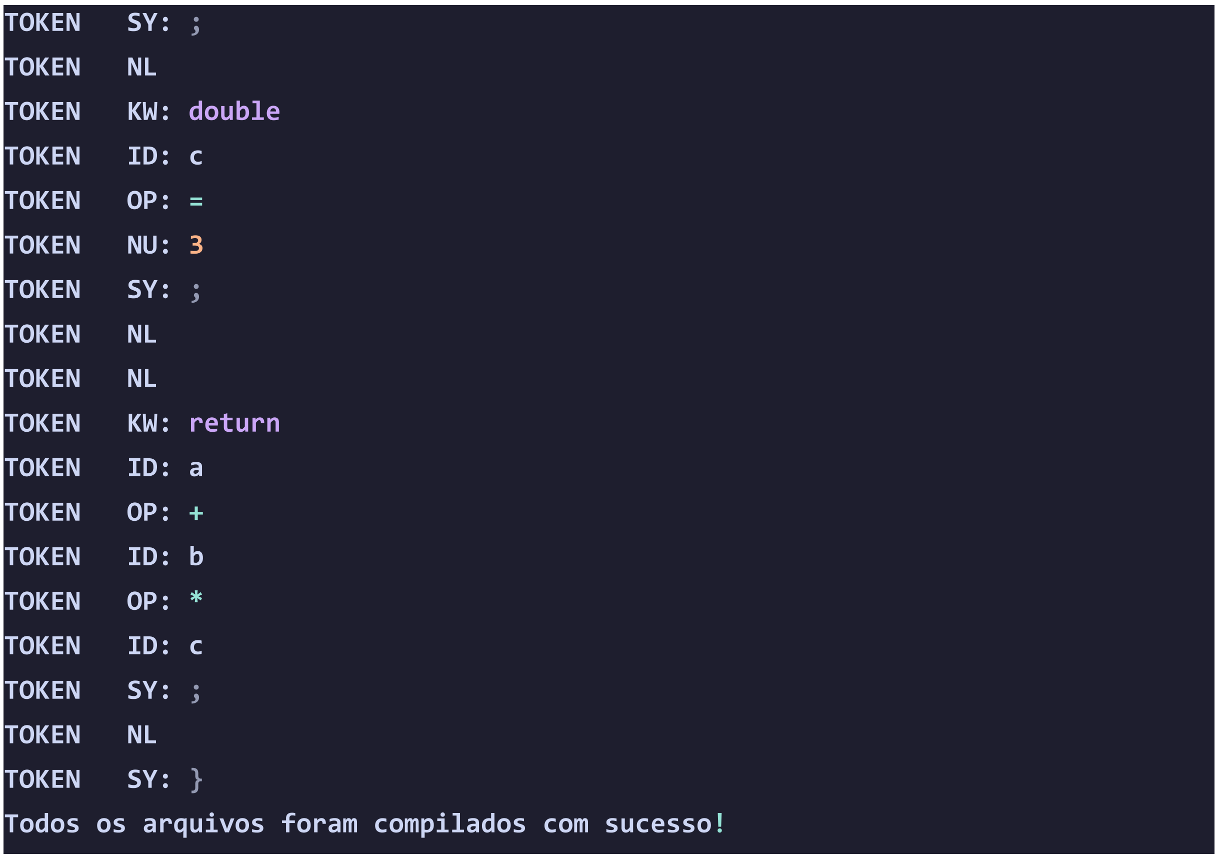


```c
#include <stdio.h>

int main()
{
    int a = 1;
    float b = 2;
    double c = 3;

    return a + b * c;
}
```

**Resultados:**

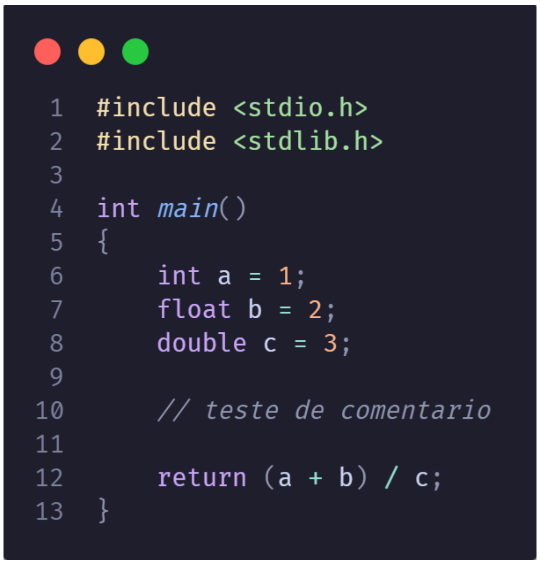


```
❯ ./main
Compiladores - TURMA A - GRUPO 7
TOKEN    SY: #
TOKEN    KW: include
TOKEN    ST: stdio.h
TOKEN    NL
TOKEN    NL
TOKEN    KW: int
TOKEN    ID: main
TOKEN    OP: (
TOKEN    SY: )
TOKEN    NL
TOKEN    SY: {
TOKEN    NL
TOKEN    KW: int
TOKEN    ID: a
TOKEN    OP: =
TOKEN    NU: 1
TOKEN    SY: ;
TOKEN    NL
TOKEN    KW: float
TOKEN    ID: b
TOKEN    OP: =
TOKEN    NU: 2
```



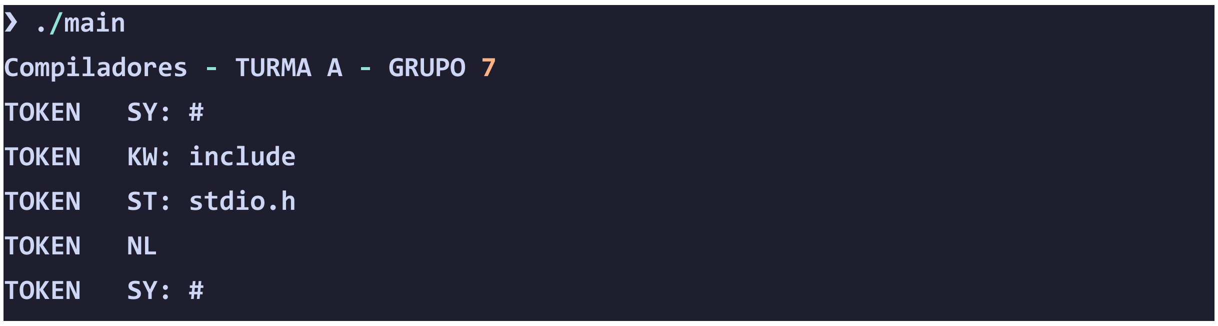
```
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: double
TOKEN   ID: c
TOKEN   OP: =
TOKEN   NU: 3
TOKEN   SY: ;
TOKEN   NL
TOKEN   NL
TOKEN   KW: return
TOKEN   ID: a
TOKEN   OP: +
TOKEN   ID: b
TOKEN   OP: *
TOKEN   ID: c
TOKEN   SY: ;
TOKEN   NL
TOKEN   SY: }
Todos os arquivos foram compilados com sucesso!
```

**c.**



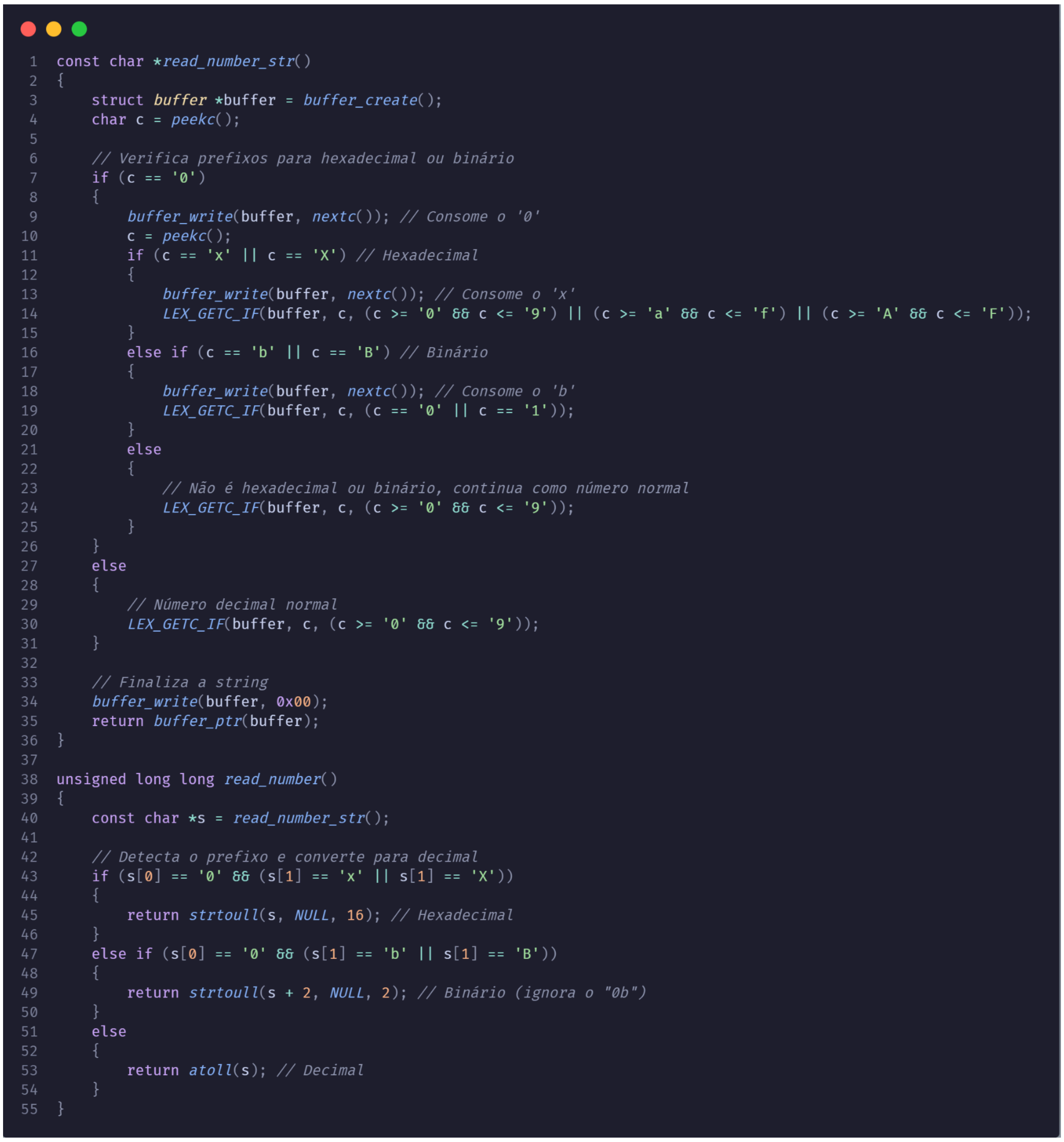
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a = 1;
    float b = 2;
    double c = 3;

    // teste de comentario

    return (a + b) / c;
}
```

**Resultados:**



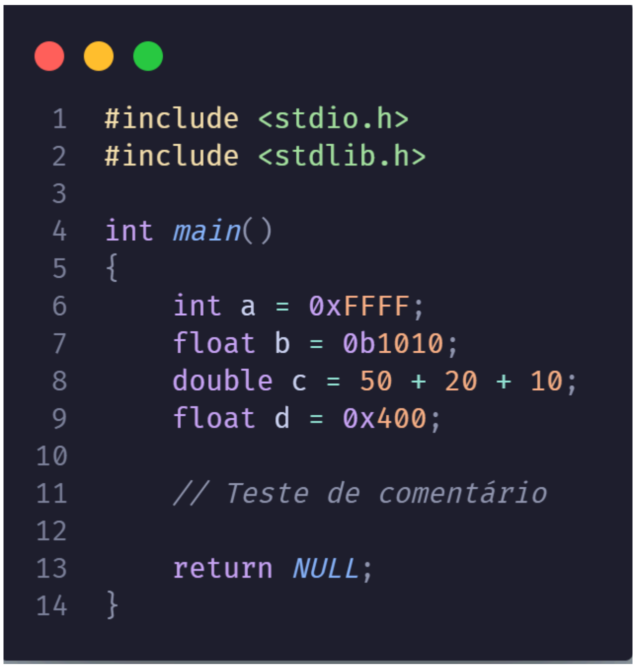
```
❯ ./main
Compiladores - TURMA A - GRUPO 7
TOKEN   SY: #
TOKEN   KW: include
TOKEN   ST: stdio.h
TOKEN   NL
TOKEN   SY: #
TOKEN   KW: include
TOKEN   ST: stdlib.h
TOKEN   NL
TOKEN   NL
TOKEN   KW: int
TOKEN   ID: main
TOKEN   OP: (
TOKEN   SY: )
TOKEN   NL
TOKEN   SY: {
TOKEN   NL
TOKEN   KW: int
TOKEN   ID: a
TOKEN   OP: =
TOKEN   NU: 1
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: float
TOKEN   ID: b
TOKEN   OP: =
TOKEN   NU: 2
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: double
TOKEN   ID: c
TOKEN   OP: =
TOKEN   NU: 3
TOKEN   SY: ;
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   KW: return
TOKEN   OP: (
TOKEN   ID: a
TOKEN   OP: +
TOKEN   ID: b
TOKEN   SY: )
TOKEN   OP: /
TOKEN   ID: c
TOKEN   SY: ;
TOKEN   NL
TOKEN    SY: }
Todos os arquivos foram compilados com sucesso!
```

3. **Implementar e converter números hexadecimais e números binários**
   **Código para converter:**



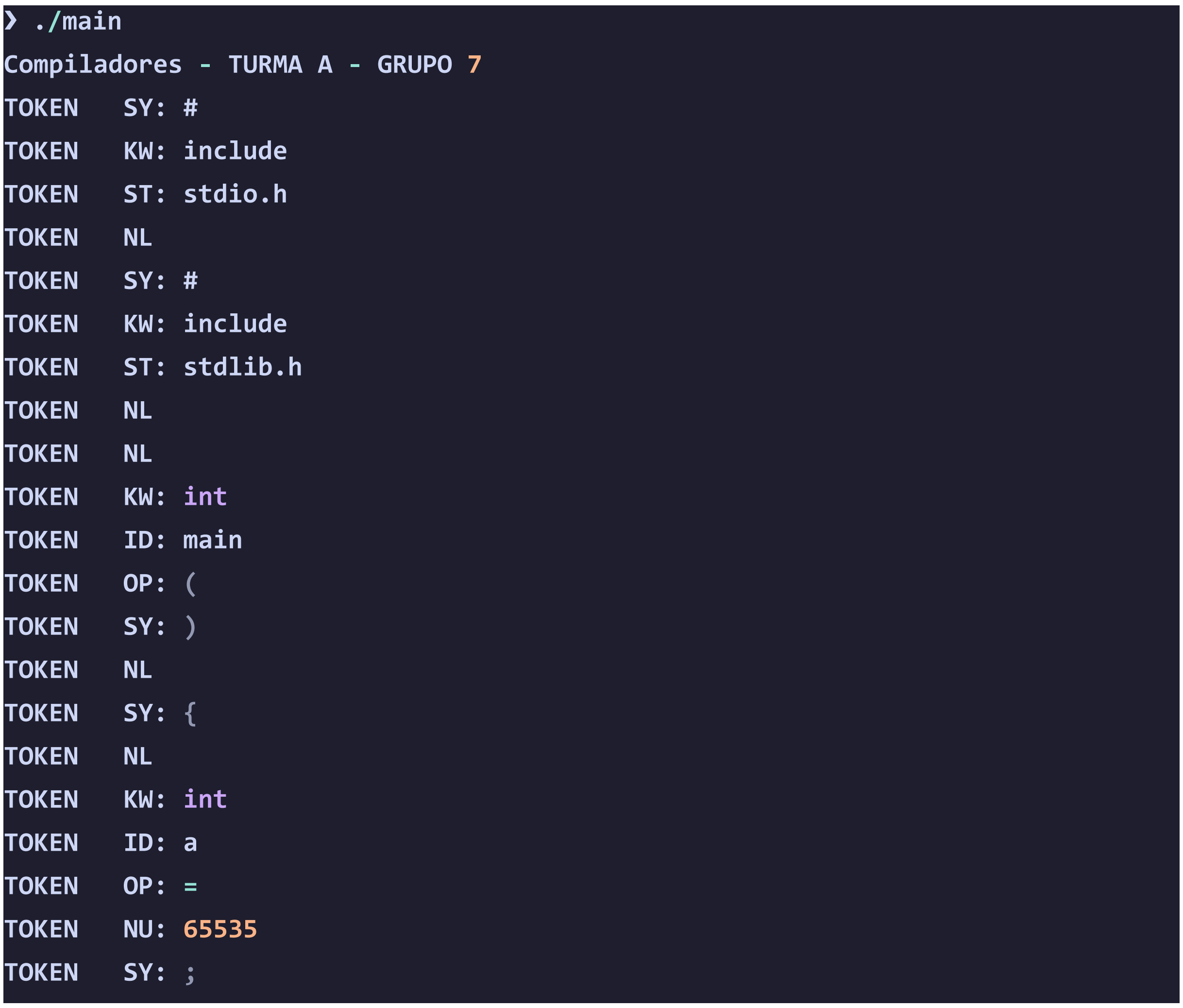
```c
const char *read_number_str()
{
    struct buffer *buffer = buffer_create();
    char c = peekc();

    // Verifica prefixos para hexadecimal ou binário
    if (c == '0')
    {
        buffer_write(buffer, nextc()); // Consome o '0'
        c = peekc();
        if (c == 'x' || c == 'X') // Hexadecimal
        {
            buffer_write(buffer, nextc()); // Consome o 'x'
            LEX_GETC_IF(buffer, c, (c >= '0' && c <= '9') || (c >= 'a' && c <= 'f') || (c >= 'A' && c <= 'F'));
        }
        else if (c == 'b' || c == 'B') // Binário
        {
            buffer_write(buffer, nextc()); // Consome o 'b'
            LEX_GETC_IF(buffer, c, (c == '0' || c == '1'));
        }
        else
        {
            // Não é hexadecimal ou binário, continua como número normal
            LEX_GETC_IF(buffer, c, (c >= '0' && c <= '9'));
        }
    }
    else
    {
        // Número decimal normal
        LEX_GETC_IF(buffer, c, (c >= '0' && c <= '9'));
    }

    // Finaliza a string
    buffer_write(buffer, 0x00);
    return buffer_ptr(buffer);
}

unsigned long long read_number()
{
    const char *s = read_number_str();

    // Detecta o prefixo e converte para decimal
    if (s[0] == '0' && (s[1] == 'x' || s[1] == 'X'))
    {
        return strtoull(s, NULL, 16); // Hexadecimal
    }
    else if (s[0] == '0' && (s[1] == 'b' || s[1] == 'B'))
    {
        return strtoull(s + 2, NULL, 2); // Binário (ignora o "0b")
    }
    else
    {
        return atoll(s); // Decimal
    }
}
```

**Teste:**

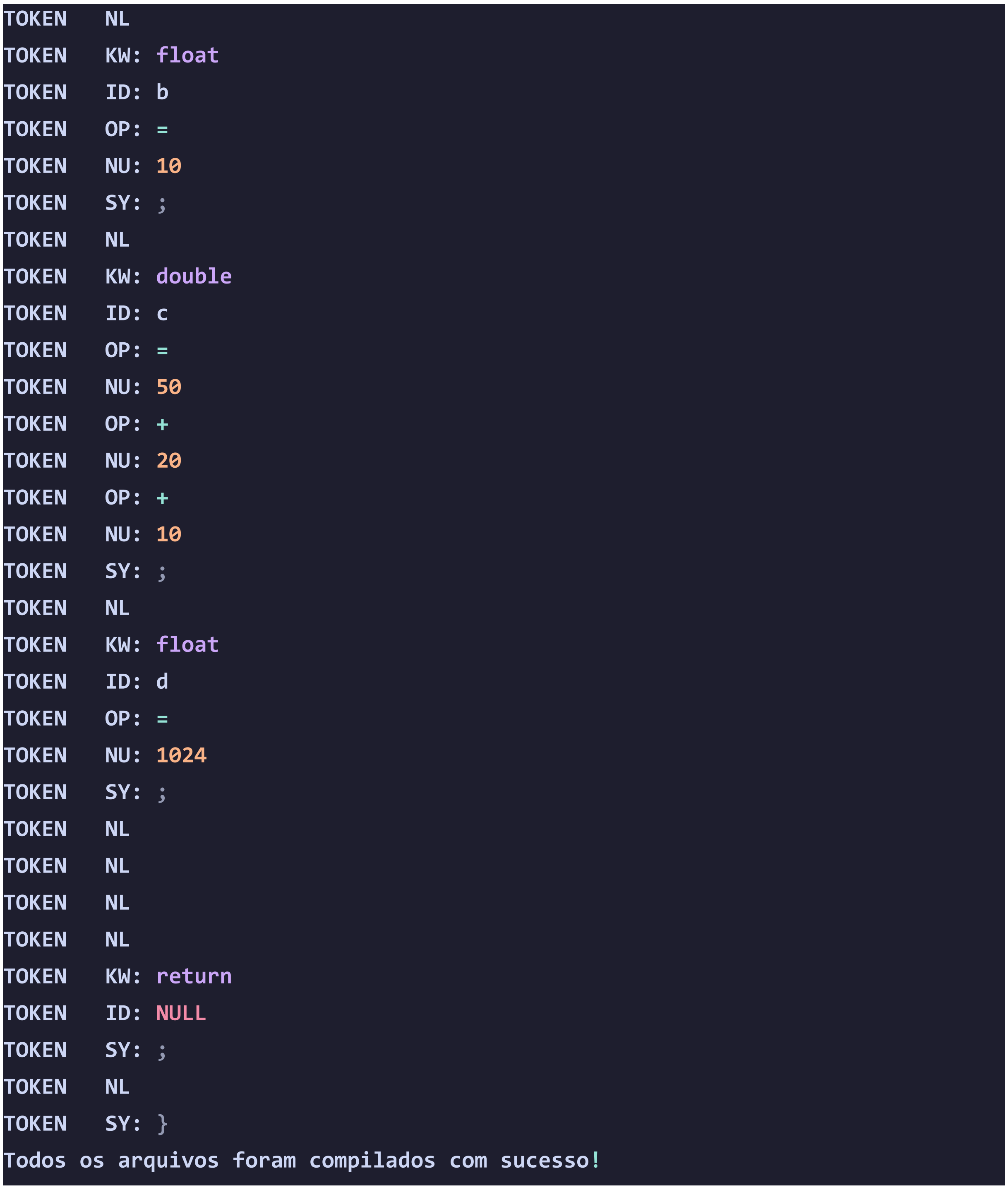


```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a = 0xFFFF;
    float b = 0b1010;
    double c = 50 + 20 + 10;
    float d = 0x400;

    // Teste de comentário

    return NULL;
}
```

**Resultados:**

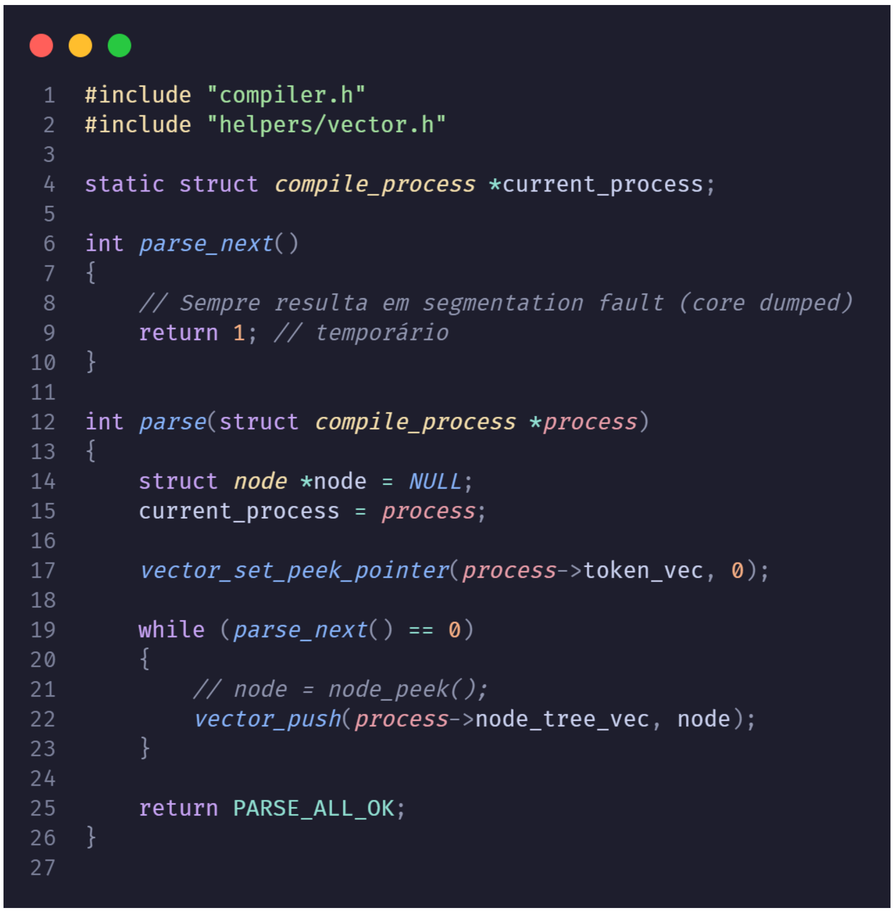


```
❯ ./main
Compiladores - TURMA A - GRUPO 7
TOKEN   SY: #
TOKEN   KW: include
TOKEN   ST: stdio.h
TOKEN   NL
TOKEN   SY: #
TOKEN   KW: include
TOKEN   ST: stdlib.h
TOKEN   NL
TOKEN   NL
TOKEN   KW: int
TOKEN   ID: main
TOKEN   OP: (
TOKEN   SY: )
TOKEN   NL
TOKEN   SY: {
TOKEN   NL
TOKEN   KW: int
TOKEN   ID: a
TOKEN   OP: =
TOKEN   NU: 65535
TOKEN   SY: ;
```



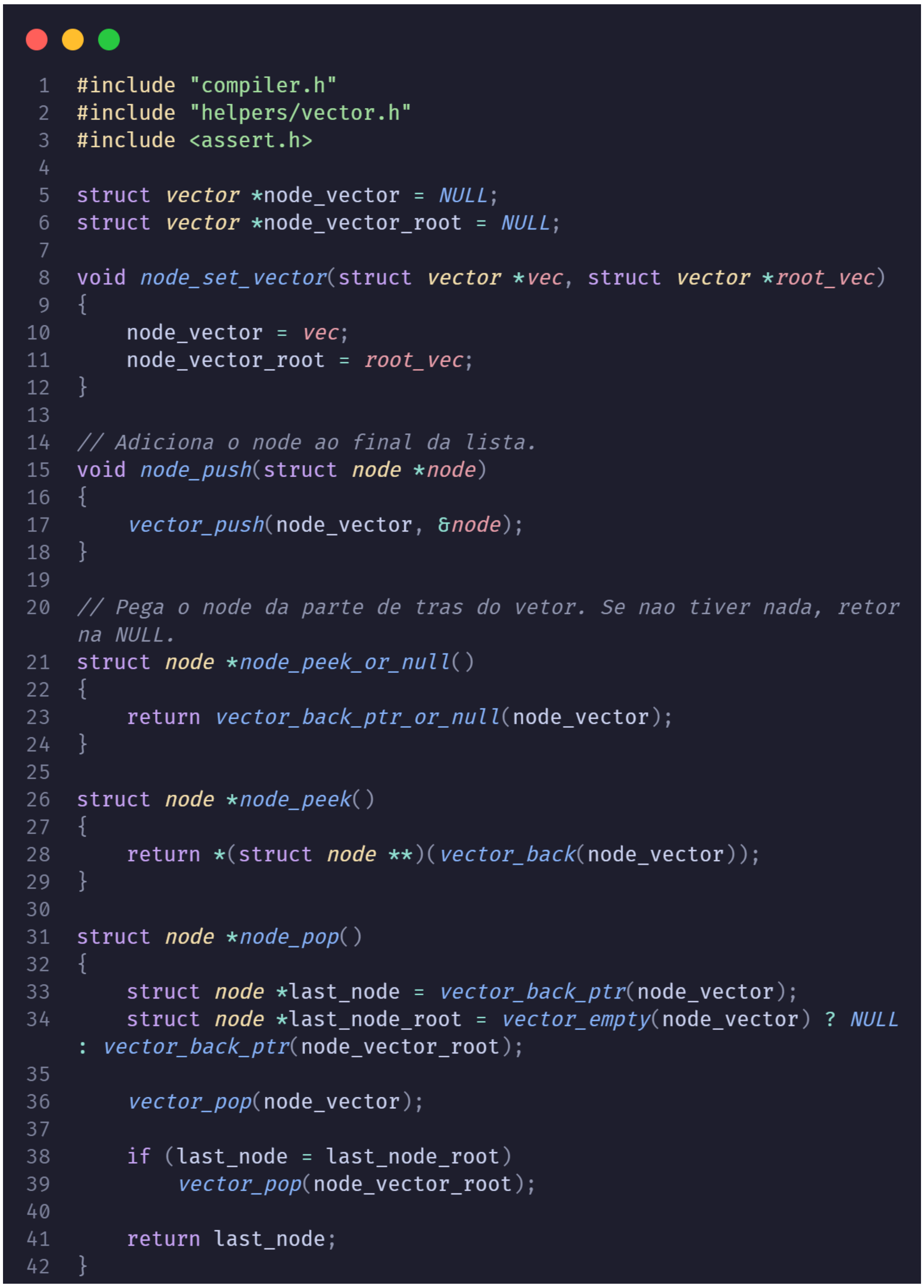
```
TOKEN   NL
TOKEN   KW: float
TOKEN   ID: b
TOKEN   OP: =
TOKEN   NU: 10
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: double
TOKEN   ID: c
TOKEN   OP: =
TOKEN   NU: 50
TOKEN   OP: +
TOKEN   NU: 20
TOKEN   OP: +
TOKEN   NU: 10
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: float
TOKEN   ID: d
TOKEN   OP: =
TOKEN   NU: 1024
TOKEN   SY: ;
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   KW: return
TOKEN   ID: NULL
TOKEN   SY: ;
TOKEN   NL
TOKEN   SY: }
Todos os arquivos foram compilados com sucesso!
```

4. **Criar os arquivos parser.c e node.c (Makefile)**
   **parse.c**



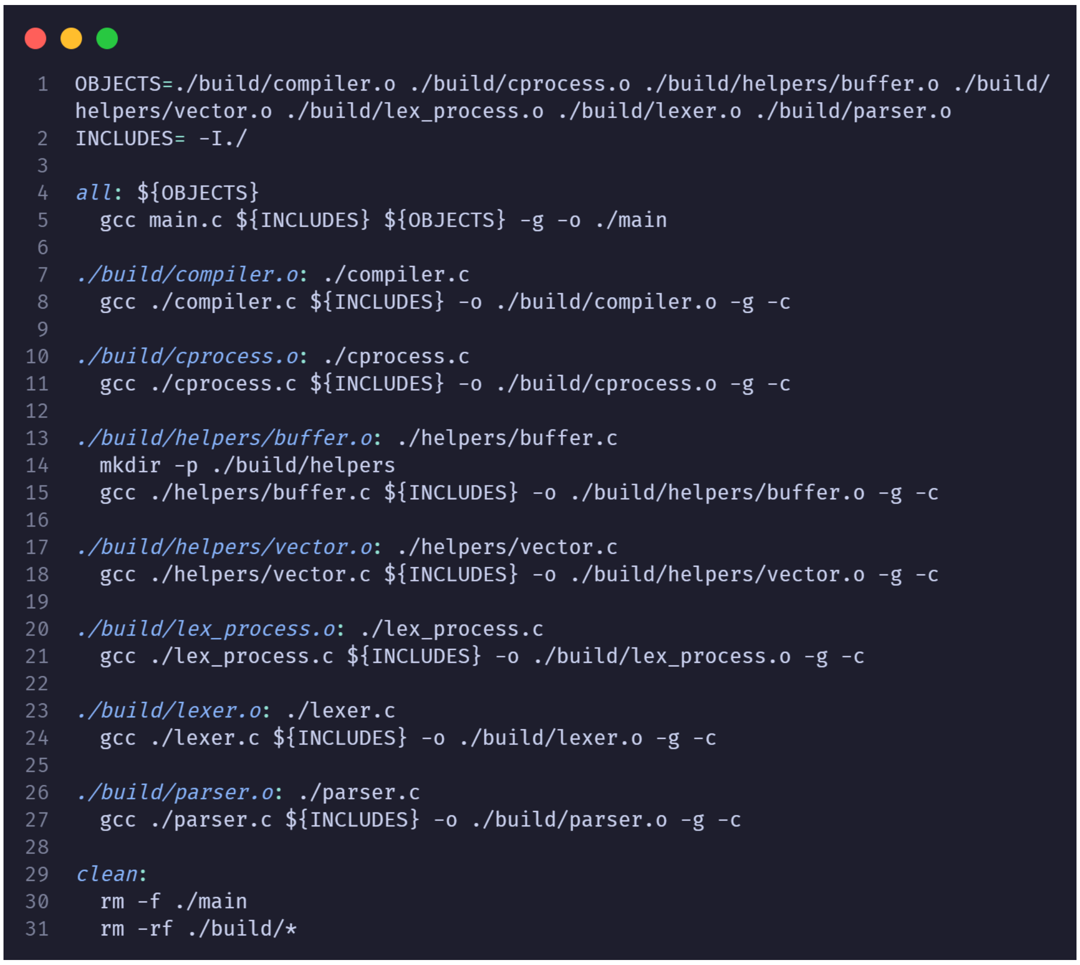
```c
#include "compiler.h"
#include "helpers/vector.h"

static struct compile_process *current_process;

int parse_next()
{
    // Sempre resulta em segmentation fault (core dumped)
    return 1; // temporário
}

int parse(struct compile_process *process)
{
    struct node *node = NULL;
    current_process = process;

    vector_set_peek_pointer(process->token_vec, 0);

    while (parse_next() == 0)
    {
        // node = node_peek();
        vector_push(process->node_tree_vec, node);
    }

    return PARSE_ALL_OK;
}

```

**node.c**



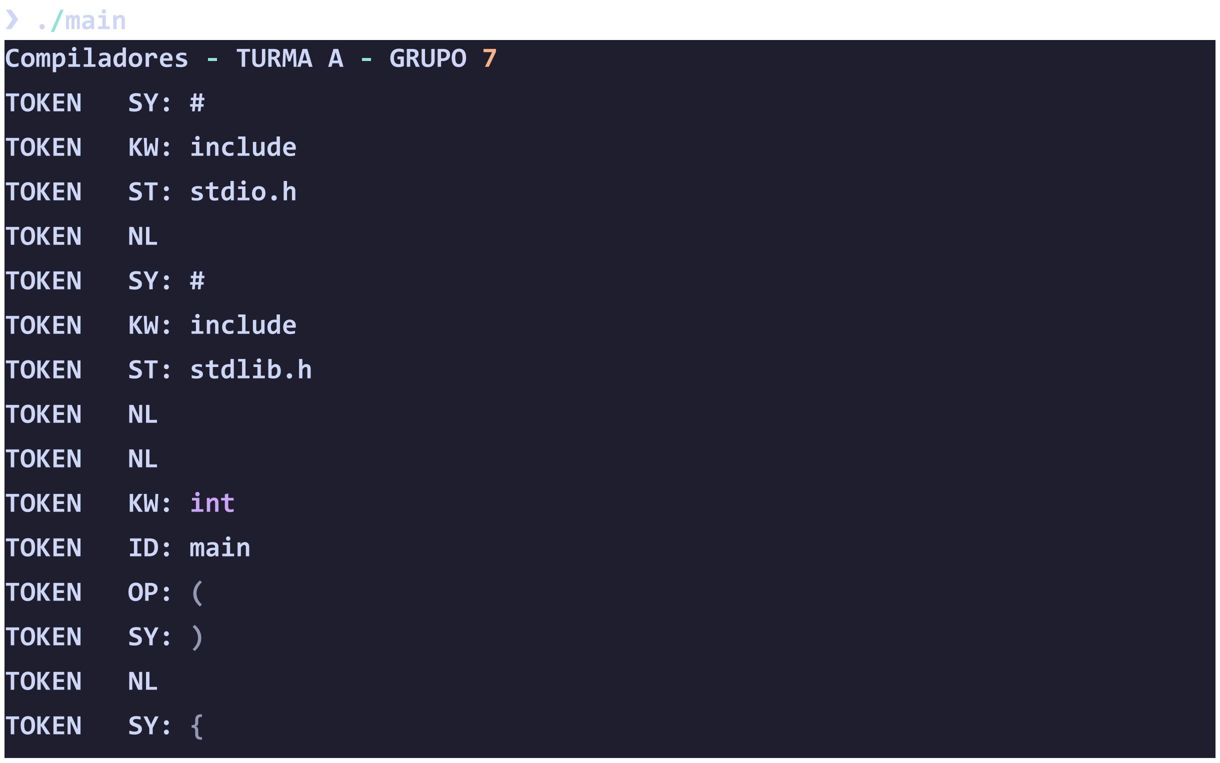
```c
#include "compiler.h"
#include "helpers/vector.h"
#include <assert.h>

struct vector *node_vector = NULL;
struct vector *node_vector_root = NULL;

void node_set_vector(struct vector *vec, struct vector *root_vec)
{
    node_vector = vec;
    node_vector_root = root_vec;
}

// Adiciona o node ao final da lista.
void node_push(struct node *node)
{
    vector_push(node_vector, &node);
}

// Pega o node da parte de tras do vetor. Se nao tiver nada, retorna NULL.
struct node *node_peek_or_null()
{
    return vector_back_ptr_or_null(node_vector);
}

struct node *node_peek()
{
    return *(struct node **)(vector_back(node_vector));
}

struct node *node_pop()
{
    struct node *last_node = vector_back_ptr(node_vector);
    struct node *last_node_root = vector_empty(node_vector) ? NULL : vector_back_ptr(node_vector_root);

    vector_pop(node_vector);

    if (last_node = last_node_root)
        vector_pop(node_vector_root);

    return last_node;
}
```

**Makefile:**

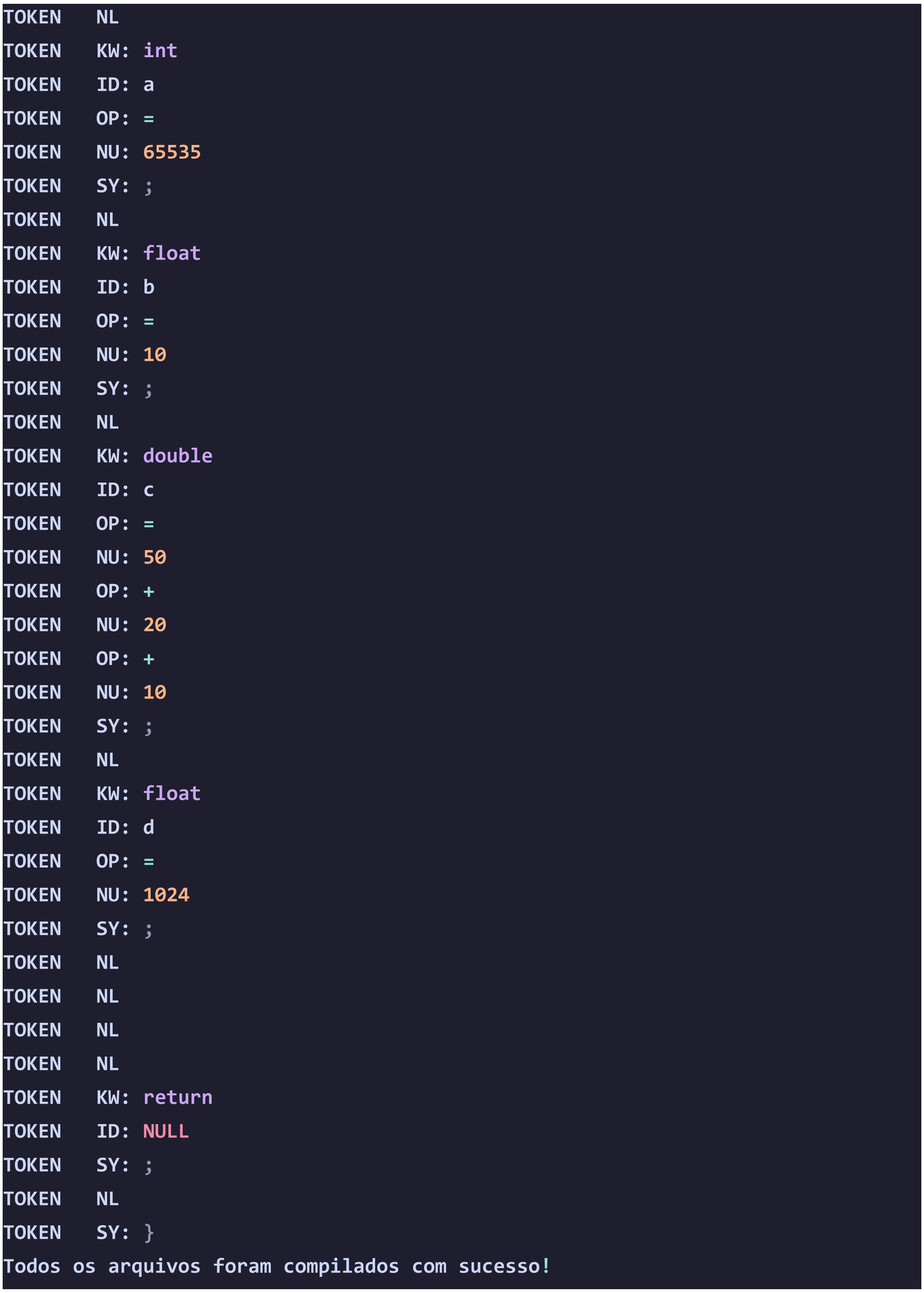


```makefile
OBJECTS=./build/compiler.o ./build/cprocess.o ./build/helpers/buffer.o ./build/helpers/vector.o ./build/lex_process.o ./build/lexer.o ./build/parser.o
INCLUDES= -I./

all: ${OBJECTS}
  gcc main.c ${INCLUDES} ${OBJECTS} -g -o ./main

./build/compiler.o: ./compiler.c
  gcc ./compiler.c ${INCLUDES} -o ./build/compiler.o -g -c

./build/cprocess.o: ./cprocess.c
  gcc ./cprocess.c ${INCLUDES} -o ./build/cprocess.o -g -c

./build/helpers/buffer.o: ./helpers/buffer.c
  mkdir -p ./build/helpers
  gcc ./helpers/buffer.c ${INCLUDES} -o ./build/helpers/buffer.o -g -c

./build/helpers/vector.o: ./helpers/vector.c
  gcc ./helpers/vector.c ${INCLUDES} -o ./build/helpers/vector.o -g -c

./build/lex_process.o: ./lex_process.c
  gcc ./lex_process.c ${INCLUDES} -o ./build/lex_process.o -g -c

./build/lexer.o: ./lexer.c
  gcc ./lexer.c ${INCLUDES} -o ./build/lexer.o -g -c

./build/parser.o: ./parser.c
  gcc ./parser.c ${INCLUDES} -o ./build/parser.o -g -c

clean:
  rm -f ./main
  rm -rf ./build/*
```

**Resultados:**



```
❯ ./main
Compiladores - TURMA A - GRUPO 7
TOKEN   SY: #
TOKEN   KW: include
TOKEN   ST: stdio.h
TOKEN   NL
TOKEN   SY: #
TOKEN   KW: include
TOKEN   ST: stdlib.h
TOKEN   NL
TOKEN   NL
TOKEN   KW: int
TOKEN   ID: main
TOKEN   OP: (
TOKEN   SY: )
TOKEN   NL
TOKEN   SY: {
```



```
TOKEN   NL
TOKEN   KW: int
TOKEN   ID: a
TOKEN   OP: =
TOKEN   NU: 65535
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: float
TOKEN   ID: b
TOKEN   OP: =
TOKEN   NU: 10
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: double
TOKEN   ID: c
TOKEN   OP: =
TOKEN   NU: 50
TOKEN   OP: +
TOKEN   NU: 20
TOKEN   OP: +
TOKEN   NU: 10
TOKEN   SY: ;
TOKEN   NL
TOKEN   KW: float
TOKEN   ID: d
TOKEN   OP: =
TOKEN   NU: 1024
TOKEN   SY: ;
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   NL
TOKEN   KW: return
TOKEN   ID: NULL
TOKEN   SY: ;
TOKEN   NL
TOKEN   SY: }
Todos os arquivos foram compilados com sucesso!
```