A white rectangular sign with red text

AI-generated content may be incorrect.

**UNIDADE CURRICULAR: Compilação**

**CÓDIGO: 21018**

**DOCENTE: Constantino Martins**

**TUTOR: Rúdi Gualter**

**Trabalho realizado pelos alunos (grupo: QUALQUER TOKEN):**

**Nome:** Andreia Romão – **Nº Estudante:** 1702430

**Nome:** Cátia Santos – **Nº Estudante:** 1702194

**Nome:** Rui Menino – **Nº Estudante:** 1103425

**Nome:** Luís Tavares – **Nº Estudante:** 1803237

**Nome:** José Augusto Azevedo – **Nº Estudante:** 2200655

**CURSO:** Licenciatura em Engenharia Informática

**DATA DE ENTREGA:** 15 de Junho de 2025

**TRABALHO / RESOLUÇÃO:**

Este relatório documenta a fase final de desenvolvimento do compilador para a linguagem MOC. O objetivo principal do projeto consistiu na criação de um compilador capaz de traduzir código fonte MOC para uma linguagem de baixo nível, tendo sido selecionado o **Assembly P3** como a arquitetura de destino.

Nesta fase final, o foco esteve na implementação do gerador de código, a última etapa do processo de compilação que ocorre após as fases de análise (léxica, sintática, semântica) e optimização da linguagem intermédia (Código de Três Endereços - TAC). Adicionalmente, foram corrigidos dois erros críticos detetados durante a implementação, um na análise semântica e outro na optimização de código, que foram cruciais para garantir a robustez e correção do compilador.

**Correções de Erros Implementadas**

No decurso da implementação, foram identificados e corrigidos dois erros significativos que afetavam a funcionalidade do compilador.

**1. Correção na Análise Semântica de acesso a Arrays**

* **Problema:** Foi detetado um erro na validação de tipos durante a análise semântica. Ao aceder a um elemento de um array do tipo double (ex: v[1]), o compilador indicava, incorretamente, que o índice do array também deveria ser do tipo double. Um índice de array deve ser sempre um valor inteiro.
* **Solução:** A regra de verificação de tipos para operações de acesso a arrays foi revista e corrigida. A nova lógica garante que o tipo de dado da expressão utilizada como índice seja sempre validado como inteiro, independentemente do tipo de dados armazenado no array. Esta correção assegura que o compilador aplica corretamente as restrições da linguagem.

**2. Correção na Otimização de Código Morto**

* **Problema:** O algoritmo de otimização, especificamente na etapa de eliminação de código morto (*dead code elimination*), apresentava uma falha na identificação do ponto de entrada do programa. O método partia do princípio de que a primeira função declarada no código era o ponto de entrada. Consequentemente, em programas onde a função main não era a primeira, esta e todas as funções por ela chamadas eram incorretamente eliminadas como código morto.
* **Solução:** Foi desenvolvido um método específico para identificar o bloco de código correspondente à função main antes de iniciar a análise de código morto. O algoritmo de optimização foi modificado para usar este bloco como a raiz da sua análise, garantindo que main e todo o código alcançável a partir dela sejam preservados.

**Implementação do Gerador de Código P3 Assembly**

A etapa final do projeto foi a criação da classe GeradorP3Assembly, responsável por traduzir a representação intermédia otimizada (Quádruplos TAC) para código Assembly P3 executável.A classe GeradorP3Assembly adopta uma abordagem de duas fases para garantir que todas as variáveis e estruturas de dados são devidamente declaradas antes da geração do código executável.

**Pré-Análise (\_pre\_scan\_quadruplos):** Antes de iniciar a tradução, o gerador realiza uma passagem completa pela lista de quádruplos. Nesta fase, identifica todos os identificadores que necessitam de alocação de memória:

* + Variáveis e temporários.
  + Arrays (através da instrução alloc), declarando-os com o pseudocódigo TAB.
  + Literais de string (usados na instrução writes), que são adicionados à secção de dados com o pseudocódigo STR. Esta análise prévia permite construir uma secção de dados completa e organizada.

**Tradução de Instruções (translate\_tac\_instruction):** Após a pré-análise, o gerador percorre novamente a lista de quádruplos e traduz cada instrução TAC para uma ou mais instruções Assembly P3.

**Mapeamento de Instruções TAC para Assembly P3**

A seguir, são detalhados alguns exemplos da lógica de tradução implementada:

* **Operações Aritméticas (+, -, \*, /, %):** Uma operação TAC como res = arg1 + arg2 é decomposta num padrão que utiliza registos. Os operandos são primeiro movidos para registos (e.g., R1, R2), a operação P3 correspondente (ADD, SUB, MUL, DIV) é executada, e o resultado é finalmente movido da localização do registo para a variável de destino. *Exemplo de tradução para res = a + b:*

Fragmento do código

MOV R1, M[a]

MOV R2, M[b]

ADD R1, R2

MOV M[res], R1

* **Operações Relacionais e Saltos (==, <, IFGOTO):** As comparações são traduzidas usando a instrução CMP do P3, que modifica os *flags* de estado. De seguida, uma instrução de salto condicional (e.g., JMP.Z para igualdade, JMP.N para menor que) é usada para desviar o fluxo de controlo. *Exemplo de tradução para if a == b goto L1:*

Fragmento do código

MOV R1, M[a]

MOV R2, M[b]

CMP R1, R2 ; ZCNO flags affected

JMP.Z L1

* **Acesso a Arrays ([] e []=):** A tradução de acesso a arrays considera a diferença entre o endereçamento em bytes (comum no TAC, onde cada elemento ocupa 4 bytes) e o endereçamento em palavras do P3 (2 bytes por palavra). O *offset* em bytes é convertido para um *offset* em palavras através de uma operação de deslocamento à direita (SHR R1, #1). O endereço final é calculado somando o endereço base do array com o *offset* em palavras, e o acesso à memória é feito através de endereçamento indireto por registo (M[R2]).
* **Chamadas de Funções (PARAM, CALL, RETURN):** O mecanismo de chamada de função foi implementado utilizando a pilha (stack):
  + PARAM arg1: O argumento é movido para um registo e depois colocado na pilha com a instrução PUSH.
  + CALL func: A instrução CALL do P3 é utilizada, que guarda o endereço de retorno na pilha e salta para a etiqueta da função.
  + RETURN val: Por convenção, o valor de retorno é colocado no registo R1 antes da instrução RET, que restaura o fluxo do programa a partir do endereço guardado na pilha.
* **Entrada e Saída (writes, writec):** As operações de escrita foram implementadas como sub-rotinas P3 reutilizáveis (WRITES, WRITEC). Quando uma instrução writes é encontrada, o gerador emite uma chamada (CALL) para a rotina WRITES, passando o endereço da string literal através da pilha. Esta abordagem modulariza o código e evita a duplicação de lógica complexa de I/O.

**Estrutura do Ficheiro Assembly de Saída**

O método final, generate\_from\_tac\_list, monta a string completa do código Assembly, organizando-a numa estrutura clara e funcional:

1. **Secção de Dados:** Iniciada no endereço 8000h (ORIG 8000h), contém todas as declarações de variáveis (WORD), arrays (TAB) e literais de string (STR). O endereço do topo da pilha (SP\_ADDRESS) é também definido aqui.
2. **Secção de Código:** Iniciada no endereço 0000h (ORIG 0000h), começa com um salto (JMP \_start) para o ponto de entrada principal.
3. **Sub-rotinas:** As funções auxiliares, como WRITES e WRITEC, são inseridas após o salto inicial.
4. **Programa Principal:** O código começa na etiqueta \_start, onde o Stack Pointer (SP) é inicializado. Segue-se o código traduzido da função main.
5. **Fim do Programa:** A execução termina com um loop infinito (Fim: BR Fim), uma prática comum para deter a execução no simulador P3, que não possui uma instrução HALT nativa.

**Conclusão**

A fase final do projeto do compilador MOC foi concluída com sucesso. A implementação do gerador de código GeradorP3Assembly demonstrou ser capaz de traduzir eficientemente a linguagem intermédia otimizada para um código Assembly P3 funcional e bem estruturado.

As correções nos módulos de análise semântica e de otimização de código foram fundamentais para aumentar a fiabilidade e o âmbito de aplicação do compilador. O projeto, no seu todo, atingiu os seus objetivos, resultando numa ferramenta de compilação robusta que cobre todas as etapas essenciais, desde a análise do código fonte até à geração de código de máquina executável.

**ANEXOS**

**Lista das Especificações da Linguagem MOC**

1. **Comentários -** Delimitadores: /\* (início) e \*/ (fim).
2. **Estrutura do Programa**
   1. **Protótipos de funções**: Devem ser declarados antes de qualquer função ou variável.
   2. **Função main()**: Ponto de entrada obrigatório.
3. **Blocos de Código -** Delimitados por {}, mesmo para blocos com uma única instrução
4. **Instruções e Operadores**
   1. **Terminação**: Todas as instruções terminam com ;.
   2. **Operadores**:
      1. Aritméticos: +, -, \*, /, %.
      2. Relacionais: ==, !=, >, <, >=, <=.
      3. Lógicos: && (E), || (OU), ! (NÃO).
      4. Atribuição: =.
   3. **Condições**: Formato restrito a Expr ou Expr OpCond Expr (e.g., x > 5 && y != 0).
5. **Estruturas de Controle**
   1. **Condicionais**:
      1. if (condicao){…} ou if(condicao){…}else{…}
      2. switch/case.
   2. **Loops**:
      1. for
      2. while.
6. **Declaração de Funções**
   1. Formato: tipo\_retorno nome(parâmetros) { ... }.
   2. Tipos de retorno: int, double, void, ou ausente (void implícito).
7. **Tipos de Dados**
   1. **Básicos**: int (inteiros), double (ponto flutuante).
   2. **Vetores**: Arrays de int ou double (e.g., int v[] = {1, 2, 3};).
   3. **Strings**: Vetores de int terminados em 0 (ASCII).
8. **Variáveis**
   1. **Declaração**:
      1. Sem inicialização: Valor padrão 0.
      2. Com inicialização: Usando expressões aritméticas (e.g., int c = 2 \* b;).
      3. Vetores: Tamanho automático se inicializados (e.g., int v[] = {1, 2};).
   2. **Âmbito**: Variáveis devem ser declaradas antes do uso.
9. **Entrada/Saída**
   1. **Entrada**:
      1. read(): Lê int ou double.
      2. readc(): Lê caracter (retorna valor ASCII).
      3. reads(): Lê string para vetor de int (termina em 0).
   2. **Saída**:
      1. write(x): Imprime valor de variável.
      2. writec(x): Imprime caracter (ASCII).
      3. writev(vetor): Imprime vetor no formato {1, 2, 0}.
      4. writes("texto"): Imprime string (com \n ao final).
10. **Conversão de Tipos**
    1. **Implícita**: int → double em operações mistas.
    2. **Explícita**: Usar (int) ou (double) (e.g., (int) 3.14 → 3).
11. **Regras Adicionais**
    1. **Strings Literais**: Usadas diretamente no writes("Olá").
    2. **Vetores como Strings**: int s[] = reads(); armazena códigos ASCII + 0.
    3. **Erros**: Uso de variáveis não declaradas é inválido.

**Erro Semântico corrigido**

**Input:**

**A black background with white text

AI-generated content may be incorrect.**

**Ouput antes da correcção:**

**A screen shot of a computer

AI-generated content may be incorrect.**

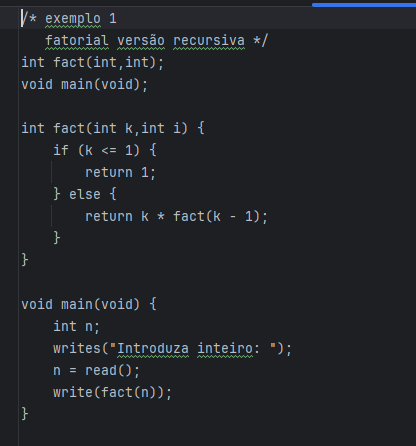
**Ouput após a correcção:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Erro Optimização corrigido**

**Input:**



**Output antes da correcção:**

python.exe main.py .\test\_examples\exemplo1.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

fact:

k = param1

i = param2

t1 = k <= 1

ifFalse t1 goto L2

L1:

return 1

goto L3

L2:

t2 = k - 1

param t2

t3 = call fact

t4 = k \* t3

return t4

L3:

end\_fact:

main:

writes "Introduza inteiro: "

t5 = call read

n = t5

param n

t6 = call fact

write t6

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

fact:

t1 = param1 <= 1

ifFalse t1 goto L2

L2:

t2 = param1 - 1

param t2

t3 = call fact

t4 = param1 \* t3

return t4

Process finished with exit code 0

**Output depois da correcção:**

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

fact:

k = param1

i = param2

t1 = k <= 1

ifFalse t1 goto L2

L1:

return 1

goto L3

L2:

t2 = k - 1

param t2

t3 = call fact

t4 = k \* t3

return t4

L3:

end\_fact:

main:

writes "Introduza inteiro: "

t5 = call read

n = t5

param n

t6 = call fact

write t6

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

fact:

t1 = param1 <= 1

ifFalse t1 goto L2

L2:

t2 = param1 - 1

param t2

t3 = call fact

t4 = param1 \* t3

return t4

main:

writes "Introduza inteiro: "

t5 = call read

param t5

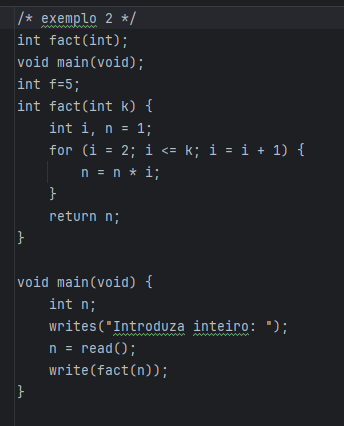
t6 = call fact

write t6

halt

end\_main:

**Input:**



**Output:**

python.exe main.py .\test\_examples\exemplo2.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

f = 5

fact:

k = param1

n = 1

i = 2

L1:

t1 = i <= k

ifFalse t1 goto L3

L2:

t2 = n \* i

n = t2

t3 = i + 1

i = t3

goto L1

L3:

return n

end\_fact:

main:

writes "Introduza inteiro: "

t4 = call read

n = t4

param n

t5 = call fact

write t5

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

fact:

n = 1

i = 2

n = 2

i = 3

L1:

t1 = i <= param1

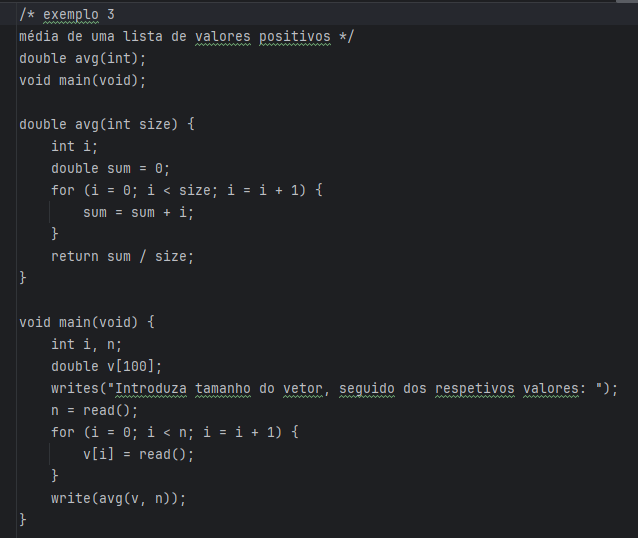
ifFalse t1 goto L3

L3:

return n

Process finished with exit code 0

**Input:**



**Output:**

python.exe main.py .\test\_examples\exemplo3.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

avg:

size = param1

sum = 0

i = 0

L1:

t1 = i < size

ifFalse t1 goto L3

L2:

t2 = sum + i

sum = t2

t3 = i + 1

i = t3

goto L1

L3:

t4 = sum / size

return t4

end\_avg:

main:

alloc v, 100

writes "Introduza tamanho do vetor, seguido dos respetivos valores: "

t5 = call read

n = t5

i = 0

L4:

t6 = i < n

ifFalse t6 goto L6

L5:

t7 = i \* 4

t8 = call read

v[t7] = t8

t9 = i + 1

i = t9

goto L4

L6:

param v

param n

t10 = call avg

write t10

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

avg:

sum = 0

i = 0

i = 1

L1:

t1 = i < param1

ifFalse t1 goto L3

L3:

t4 = sum / param1

return t4

Process finished with exit code 0

**Input:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\exemplo4.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

x = 10

write x

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

write 10

halt

end\_main:

Process finished with exit code 0

**Input:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\exemplo5.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

x = 10

y = x

t1 = x == 10

ifFalse t1 goto L2

L1:

writes "10"

goto L3

L2:

t2 = x + 25

x = t2

L3:

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

x = 10

L1:

writes "10"

goto L3

L3:

halt

end\_main:

Process finished with exit code 0

**Input:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\exemplo6.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

a = 10

t1 = b + 10

c = t1

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

t1 = b + 10

halt

end\_main:

Process finished with exit code 0

**Input:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\exemplo7.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

x = 10

y = 20

t1 = call read

n = t1

i = 0

L1:

t2 = i < n

ifFalse t2 goto L3

L2:

t3 = x + y

z = t3

write z

t4 = i + 1

i = t4

goto L1

L3:

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

t1 = call read

i = 0

i = 1

L1:

t2 = i < t1

ifFalse t2 goto L3

L3:

halt

end\_main:

Process finished with exit code 0

**Input:**

A screenshot of a computer

AI-generated content may be incorrect.

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Input:**

A screen shot of a computer

AI-generated content may be incorrect.

**Output:**

A computer screen shot of a program

AI-generated content may be incorrect.

**Input:**

A computer screen with white text

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\teste\_ciclo.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

soma = 0

i = 0

L1:

t1 = i < 10

ifFalse t1 goto L3

L2:

t2 = i == 5

ifFalse t2 goto L5

L4:

write i

L5:

t3 = soma + i

soma = t3

write soma

t4 = i + 1

goto L1

L3:

write soma

return 0

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

soma = 0

t2 = False

t3 = 0

L1:

L2:

ifFalse t2 goto L5

L5:

soma = t3

write soma

goto L1

Process finished with exit code 0

**Input:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

A computer screen with white text

AI-generated content may be incorrect.

**Input:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .c\test\_examples\teste\_geral.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

t1 = 2 + 3

x = t1

y = x

z = y

t2 = 10 - 4

r = t2

t = r

a = 7

b = 3

t3 = a + b

soma = t3

t4 = (double) soma

d = t4

alloc v, 3

t5 = 0 \* 4

v[t5] = 1

t6 = 1 \* 4

v[t6] = 2

t7 = 2 \* 4

v[t7] = 3

i = 0

total = 0

L1:

t8 = i < 3

ifFalse t8 goto L3

L2:

t9 = i \* 4

t10 = v[t9]

t11 = total + t10

total = t11

t12 = i + 1

i = t12

goto L1

L3:

t13 = total > 5

ifFalse t13 goto L5

L4:

write total

goto L6

L5:

write 0

L6:

copia = total

resultado = copia

write resultado

t14 = call read

t15 = t14

t16 = call readc

t17 = t16

t18 = call reads

t19 = t18

writes "fim"

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

t4 = (double) 10

alloc v, 3

v[0] = 1

v[4] = 2

v[8] = 3

total = 0

t9 = 0

L1:

L2:

t10 = v[t9]

t11 = total + t10

total = t11

goto L1

Process finished with exit code 0

**Input:**

A computer screen shot of a program

AI-generated content may be incorrect.

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Input:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador01.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

x = 5

a = 2.5

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

halt

end\_main:

Process finished with exit code 0

**Input:**

A computer screen shot of a black screen

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador02.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

t1 = 0 \* 4

x[t1] = 1

t2 = 1 \* 4

x[t2] = 2

t3 = 2 \* 4

x[t3] = 3

t4 = 0 \* 4

t5 = x[t4]

t6 = 1 \* 4

t7 = x[t6]

t8 = t5 + t7

y = t8

write y

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

x[0] = 1

x[4] = 2

x[8] = 3

t4 = 0

t5 = x[t4]

t7 = x[4]

t8 = t5 + t7

write t8

halt

end\_main:

Process finished with exit code 0

**Input:**

A screen shot of a computer

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador03.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

a = 5

b = 10

t1 = a + b

t2 = t1 \* 2

c = t2

t3 = c / 5.0

d = t3

write d

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

write 6.0

halt

end\_main:

Process finished with exit code 0

**Input:**

A screen shot of a computer

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador04.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

a = 50

b = 10

t1 = a + b

c = t1

t2 = 3 \* b

d = t2

t3 = c - d

e = t3

write e

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

write 30

halt

end\_main:

Process finished with exit code 0

**Input:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador05.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

a = 5

b = 10

t1 = b \* 2

t2 = a + t1

c = t2

t3 = b + 1

d = t3

write d

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

write 11

halt

end\_main:

Process finished with exit code 0

**Input:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador06.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

a = 5

b = 10

t1 = b \* 2.0

t2 = a + t1

c = t2

t3 = c / 5.0

d = t3

write d

t4 = call read

f = t4

t5 = 2 \* b

g = t5

t6 = 0 \* 4

x[t6] = 1

t7 = 1 \* 4

x[t7] = 2

t8 = 2 \* 4

x[t8] = 3

t9 = 0 \* 4

t10 = x[t9]

t11 = 1 \* 4

t12 = x[t11]

t13 = t10 + t12

y = t13

write y

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

write 5.0

t4 = call read

x[0] = 1

x[4] = 2

x[8] = 3

t9 = 0

t10 = x[t9]

t12 = x[4]

t13 = t10 + t12

write t13

halt

end\_main:

Process finished with exit code 0

**Input:**

A computer screen with white text

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador07.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

x = 15

y = x

t1 = x == 10

ifFalse t1 goto L2

L1:

writes "10"

goto L3

L2:

t2 = x + 25

x = t2

L3:

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

x = 15

t1 = False

ifFalse t1 goto L2

L2:

x = 40

L3:

halt

end\_main:

Process finished with exit code 0

**Input:**

A computer screen with white text and numbers

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador08.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

x = 10

y = x

t1 = x == 10

ifFalse t1 goto L2

L1:

writes "10"

goto L3

L2:

t2 = x + 25

x = t2

L3:

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

x = 10

L1:

writes "10"

goto L3

L3:

halt

end\_main:

Process finished with exit code 0

**Input:**

A screen shot of a computer program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador09.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

obter\_valor:

valor = 42

return valor

t1 = valor + 10

valor = t1

write valor

end\_obter\_valor:

main:

t2 = call obter\_valor

resultado = t2

write resultado

return 0

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

obter\_valor:

valor = 42

return valor

Process finished with exit code 0

**Input:**

A computer screen shot of a program

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador10.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

soma = 0

i = 0

L1:

t1 = i < 10

ifFalse t1 goto L3

L2:

t2 = i == 5

ifFalse t2 goto L5

L4:

write i

L5:

t3 = soma + i

soma = t3

write soma

t4 = i + 1

goto L1

L3:

write soma

return 0

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

soma = 0

t2 = False

t3 = 0

L1:

L2:

ifFalse t2 goto L5

L5:

soma = t3

write soma

goto L1

Process finished with exit code 0

**Input:**

A computer screen shot of a computer code

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador11.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

a = 10

t1 = b + 10

c = t1

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

t1 = b + 10

halt

end\_main:

Process finished with exit code 0

**Input:**

A screen shot of a computer

AI-generated content may be incorrect.

**Output:**

python.exe main.py .\test\_examples\Testes\_optimizador12.moc

--- A iniciar Análise Sintática ---

--- Análise Sintática concluída ---

--- A iniciar Análise Semântica ---

--- Análise Semântica concluída ---

--- A iniciar Geração de Código Intermédio ---

--- Geração de Código Intermédio concluída ---

==== CÓDIGO TAC GERADO ====

main:

a = 100

write a

b = a

a = 10

t1 = b + 20

c = t1

write c

halt

end\_main:

==== CÓDIGO TAC OTIMIZADO ====

main:

a = 100

write a

a = 10

write 120

halt

end\_main:

Process finished with exit code 0

**Bibliografia/Referências:**

* Compilers: principles, techniques and tools, 2nd Ed., Aho, Lam, Setti, Ullman, Addison-Wesley, 2007
* Compiladores – Da Teoria à Prática, Pedro Reis Santos e Thibault Langlois. FCA, 2015.
* The ANTLR Mega Tutorial: <https://tomassetti.me/antlr-mega-tutorial/>
* ANTLR Doc: <https://github.com/antlr/antlr4/blob/master/doc/index.md>
* CD | INTRODUCTION | INTRODUCTION AND VARIOUS PHASES OF COMPILER | RAVINDRABABU RAVULA

<https://youtu.be/Qkwj65l_96I?list=PL5UbMb0H_A9hs6Z_myVW_tqRpFipkzniD>

* EECS4302 ANTLR4 PARSER GENERATOR TUTORIAL

<https://youtu.be/-FdD_xzNFL4?list=PL5UbMb0H_A9hs6Z_myVW_tqRpFipkzniD>

* COMPILADORES DE JUDSON SANTIAGO

<https://www.youtube.com/playlist?list=PLX6Nyaq0ebfhI396WlWN6WlBm-tp7vDtV>

* COMPILADORES DE PROF. JOSÉ RUI

<https://www.youtube.com/playlist?list=PLqlIQgAFrQ14VmHe8VbIVUkBv5Hziv86->

* COMPILADORES (CC3001) — 2022/2023 — Professor Pedro Vasconcelos, 2022.

Faculdade de Ciências da Universidade do Porto

<https://www.dcc.fc.up.pt/~pbv/aulas/compiladores/teoricas/>