Compiladores

– Linguagem COOL / Projeto de Curso –

Matheus Alcântara Souza

Engenharia de Computação – PUC Minas

Belo Horizonte, Brasil

2022

Sumário

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

COOL ≡ *Classroom Object Oriented Language*

Desenvolvida para

- Ser implementada em pouco tempo
- Fornecer uma amostra de uma implementação moderna
  - Abstração
  - Reuso (por meio de herança)
  - Gerenciamento de memória
  - e mais ...

Pórem muitos conceitos e construções foram deixados de fora ☹

Introdução à Linguagem COOL

Exemplo Simples

```
class Point {
    x : Int <- 0;
    y : Int <- 0;
};
```

Programas em COOL são conjunto de definições de classes

- Deve existir uma classe **Main** com um método **main**
- Não há a noção separada de subrotinas

⇒ Classe em COOL ≡ coleção de atributos e métodos

⇒ Instâncias de uma classe são objetos

Introdução à Linguagem COOL

Exemplo Simples

```
class Point {
    x : Int <- 0;
    y : Int <- 0;
};
```

Programas em COOL são conjunto de definições de classes

- Deve existir uma classe **Main** com um método **main**
- Não há a noção separada de subrotinas

⇒ Classe em COOL = coleção de atributos e métodos

⇒ Instâncias de uma classe são objetos

Introdução à Linguagem COOL

Exemplo Simples

```
class Point {
    x : Int <- 0;
    y : Int <- 0;
};
```

Programas em COOL são conjunto de definições de classes

- Deve existir uma classe **Main** com um método **main**
- Não há a noção separada de subrotinas

⇒ Classe em COOL ≡ coleção de atributos e métodos

⇒ Instâncias de uma classe são objetos

Introdução à Linguagem COOL

Exemplo Simples

```
class Point {
    x : Int <- 0;
    y : Int <- 0;
};
```

Programas em COOL são conjunto de definições de classes

- Deve existir uma classe **Main** com um método **main**
- Não há a noção separada de subrotinas

⇒ Classe em COOL ≡ coleção de atributos e métodos

⇒ Instâncias de uma classe são objetos

Introdução à Linguagem COOL

Exemplo Simples

```
class Point {
    x : Int <- 0;
    y : Int <- 0;
};
```

Programas em COOL são conjunto de definições de classes

- Deve existir uma classe **Main** com um método **main**
- Não há a noção separada de subrotinas

$\Rightarrow$ Classe em COOL $\equiv$ coleção de atributos e métodos

$\Rightarrow$ Instâncias de uma classe são objetos

Introdução à Linguagem COOL

Exemplo Simples

```
class Point {
    x : Int <- 0;
    y : Int <- 0;
};
```

Programas em COOL são conjunto de definições de classes

- Deve existir uma classe **Main** com um método **main**
- Não há a noção separada de subrotinas

$\Rightarrow$ Classe em COOL $\equiv$ coleção de atributos e métodos

$\Rightarrow$ Instâncias de uma classe são objetos

Introdução à Linguagem COOL

Objetos em COOL

```
class Point {
    x : Int <- 0;
    y : Int;
};
```

A expressão `new Point` cria uma nova instância (ou objeto) da classe `Point`

Um objeto pode ser "entendido" de forma simplificada como um registro com uma entrada para cada atributo

Introdução à Linguagem COOL

Objetos em COOL

```
class Point {
    x : Int <- 0;
    y : Int; (* usa valor default *)
};
```

A expressão `new Point` cria uma nova instância (ou objeto) da classe `Point`

Um objeto pode ser "entendido" de forma simplificada como um registro com uma entrada para cada atributo

Introdução à Linguagem COOL

Objetos em COOL

```
class Point {
    x : Int <- 0;
    y : Int; (* usa valor default *)
};
```

A expressão `new Point` cria uma nova instância (ou objeto) da classe `Point`

Um objeto pode ser "entendido" de forma simplificada como um registro com uma entrada para cada atributo

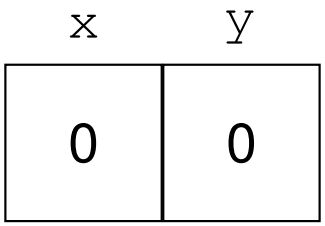
Introdução à Linguagem COOL

Objetos em COOL

```
class Point {
    x : Int <- 0;
    y : Int; (* usa valor default *)
};
```

A expressão `new Point` cria uma nova instância (ou objeto) da classe `Point`

Um objeto pode ser "entendido" de forma simplificada como um registro com uma entrada para cada atributo



Introdução à Linguagem COOL

Métodos em COOL

Além de atributos, uma classe também pode definir métodos para manipulá-los

```
class Point {
   x : Int <- 0;
   y : Int; (* usa valor default *)

   movePoint(newx : Int, newy : Int): Point {
      { x <- newx;
        y <- newy;
        self;
      } -- termina a expressão em bloco
   }; -- termina o método
}; -- termina a classe
```

Métodos podem se referir ao objeto corrente por meio de `self`

Introdução à Linguagem COOL

Métodos em COOL

Além de atributos, uma classe também pode definir métodos para manipulá-los

```
class Point {
   x :  Int <- 0;
   y :  Int; (* usa valor default *)

   movePoint(newx :  Int, newy :  Int):  Point {
      { x <- newx;
        y <- newy;
        self;
      } -- termina a expressão em bloco
   }; -- termina o método
}; -- termina a classe
```

Métodos podem se referir ao objeto corrente por meio de `self`

Introdução à Linguagem COOL

Métodos em COOL

Além de atributos, uma classe também pode definir métodos para manipulá-los

```
class Point {
   x :  Int <- 0;
   y :  Int; (* usa valor default *)

   movePoint(newx :  Int, newy :  Int):  Point {
      { x <- newx;
        y <- newy;
        self;
      } -- termina a expressão em bloco
   }; -- termina o método
}; -- termina a classe
```

Métodos podem se referir ao objeto corrente por meio de `self`

Introdução à Linguagem COOL

Encapsulamento e Controle de Visibilidade em COOL

Todos os métodos são globais (e públicos)

Atributos são locais (privados) a uma classe

- Eles só podem ser acessados pelos métodos da própria classe

```
class Point {
    ⋮
    x(): Int { x };
    setx(newx : Int): Int { x <- newx };
};
```

⇒ Quando necessário, deve-se implementar *getters* e *setters*

Introdução à Linguagem COOL

Encapsulamento e Controle de Visibilidade em COOL

Todos os métodos são globais (e públicos)

Atributos são locais (privados) a uma classe

- Eles só podem ser acessados pelos métodos da própria classe

```
class Point {
    ⋮
    x(): Int { x };
    setx(newx : Int): Int { x <- newx };
};
```

⇒ Quando necessário, deve-se implementar *getters* e *setters*

Introdução à Linguagem COOL

Encapsulamento e Controle de Visibilidade em COOL

Todos os métodos são globais (e públicos)

Atributos são locais (privados) a uma classe

- Eles só podem ser acessados pelos métodos da própria classe

```
class Point {
    ⋮
    x(): Int { x };
    setx(newx : Int): Int { x <- newx };
};
```

⇒ Quando necessário, deve-se implementar *getters* e *setters*

Introdução à Linguagem COOL

Encapsulamento e Controle de Visibilidade em COOL

Todos os métodos são globais (e públicos)

Atributos são locais (privados) a uma classe

- Eles só podem ser acessados pelos métodos da própria classe

```
class Point {
    ⋮
    x(): Int { x };
    setx(newx : Int): Int { x <- newx };
};
```

⇒ Quando necessário, deve-se implementar *getters* e *setters*

Introdução à Linguagem COOL

Métodos em COOL (cont.)

Cada objeto “sabe” como acessar o código de um dado método

Deve-se imaginar que o objeto contém um “*slot*” que aponta para o código

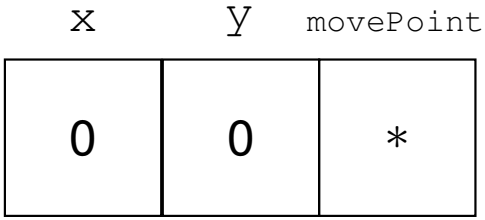
Na realidade, implementações economizam espação compartilhando esse ponteiros entre instâncias/objetos de uma mesma classe

Introdução à Linguagem COOL

Métodos em COOL (cont.)

Cada objeto “sabe” como acessar o código de um dado método

Deve-se imaginar que o objeto contém um “*slot*” que aponta para o código



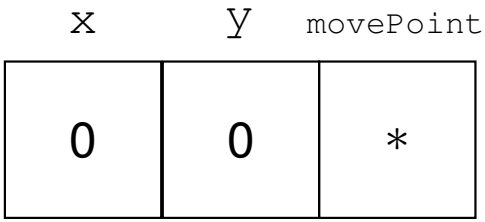
Na realidade, implementações economizam espaço compartilhando esse ponteiros entre instâncias/objetos de uma mesma classe

Introdução à Linguagem COOL

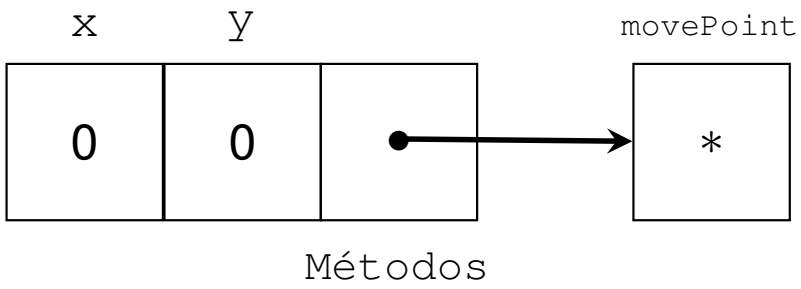
Métodos em COOL (cont.)

Cada objeto “sabe” como acessar o código de um dado método

Deve-se imaginar que o objeto contém um “*slot*” que aponta para o código



Na realidade, implementações economizam espação compartilhando esse ponteiros entre instâncias/objetos de uma mesma classe



Introdução à Linguagem COOL

Herança em COOL

Uma classe pode ser estendida criando uma hierarquia de classes

```
class ColorPoint inherits Point {
   color : Int <- 0;
   movePoint(newx : Int, newy : Int): Point {
      { color <- 0;
        x <- newx;
        y <- newy;
        self;
      }
   };
};
```

Introdução à Linguagem COOL

Herança em COOL

Uma classe pode ser estendida criando uma hierarquia de classes

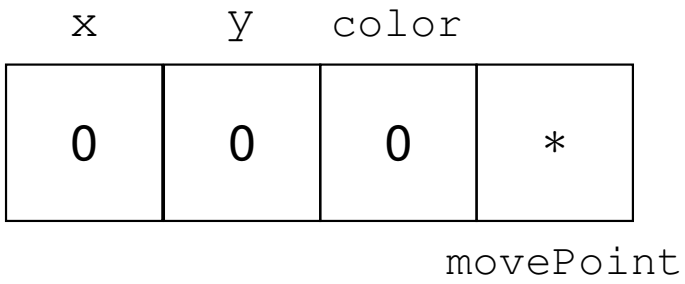
```
class ColorPoint inherits Point {
   color : Int <- 0;
   movePoint(newx : Int, newy : Int): Point {
      { color <- 0;
        x <- newx;
        y <- newy;
        self;
      }
   };
};
```

Introdução à Linguagem COOL

Herança em COOL

Uma classe pode ser estendida criando uma hierarquia de classes

```
class ColorPoint inherits Point {
   color : Int <- 0;
   movePoint(newx : Int, newy : Int): Point {
      { color <- 0;
        x <- newx;
        y <- newy;
        self;
      }
   };
};
```



Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Tipos em COOL

Toda classe representa um tipo

Tipos básicos

- `Int`: para representar inteiros
- `Bool`: para representar valores booleanos → `true` ou `false`
- `String`: para representar cadeias de caracteres
- `Object`: para representar a raiz da hierarquia de classes

Todas as variáveis devem ser declaradas antes do uso

- O compilador de COOL deve inferir tipos para expressões

Introdução à Linguagem COOL

Verificação de Tipos em COOL

```
x : A;
x <- new B;
```

A declaração acima é correta se `A` é um ancestral de `B` na hierarquia de classes

- Em qualquer lugar em que um objeto da classe `A` seja esperada um objeto da classe `B` pode ser usado

Type safety

- Um programa "bem-tipado" não pode resultar em erros de tipo durante sua execução

Introdução à Linguagem COOL

Verificação de Tipos em COOL

```
x : A;
x <- new B;
```

A declaração acima é correta se `A` é um ancestral de `B` na hierarquia de classes

- Em qualquer lugar em que um objeto da classe `A` seja esperada um objeto da classe `B` pode ser usado

Type safety

- Um programa "bem-tipado" não pode resultar em erros de tipo durante sua execução

Introdução à Linguagem COOL

Verificação de Tipos em COOL

```
x : A;
x <- new B;
```

A declaração acima é correta se `A` é um ancestral de `B` na hierarquia de classes

- Em qualquer lugar em que um objeto da classe `A` seja esperada um objeto da classe `B` pode ser usado

Type safety

- Um programa "bem-tipado" não pode resultar em erros de tipo durante sua execução

Introdução à Linguagem COOL

Verificação de Tipos em COOL

```
x : A;
x <- new B;
```

A declaração acima é correta se `A` é um ancestral de `B` na hierarquia de classes

- Em qualquer lugar em que um objeto da classe `A` seja esperada um objeto da classe `B` pode ser usado

Type safety

- Um programa "bem-tipado" não pode resultar em erros de tipo durante sua execução

Introdução à Linguagem COOL

Verificação de Tipos em COOL

```
x : A;
x <- new B;
```

A declaração acima é correta se `A` é um ancestral de `B` na hierarquia de classes

- Em qualquer lugar em que um objeto da classe `A` seja esperada um objeto da classe `B` pode ser usado

Type safety

- Um programa “bem-tipado” não pode resultar em erros de tipo durante sua execução

Introdução à Linguagem COOL

Invocação de Métodos $\times$ Herança

Métodos são invocados por despacho

Compreender o despacho na presença de herança é um aspecto sutil das linguagens OO, por exemplo

```
p : Point;
p <- new ColorPoint;
p.movePoint(1,2);
```

- O tipo estático de `p` é a classe `Point`
- Já o tipo dinâmico de `p` é a classe `ColorPoint`
- Dessa forma, a chamada `p.movePoint` deve invocar a versão da classe `ColorPoint`

Introdução à Linguagem COOL

Invocação de Métodos $\times$ Herança

Métodos são invocados por despacho

Compreender o despacho na presença de herança é um aspecto sutil das linguagens OO, por exemplo

```
p : Point;
p <- new ColorPoint;
p.movePoint(1,2);
```

- O tipo estático de `p` é a classe `Point`
- Já o tipo dinâmico de `p` é a classe `ColorPoint`
- Dessa forma, a chamada `p.movePoint` deve invocar a versão da classe `ColorPoint`

Introdução à Linguagem COOL

Invocação de Métodos × Herança

Métodos são invocados por despacho

Compreender o despacho na presença de herança é um aspecto sutil das linguagens OO, por exemplo

```
p : Point;
p <- new ColorPoint;
p.movePoint(1,2);
```

- O tipo estático de `p` é a classe `Point`
- Já o tipo dinâmico de `p` é a classe `ColorPoint`
- Dessa forma, a chamada `p.movePoint` deve invocar a versão da classe `ColorPoint`

Introdução à Linguagem COOL

Invocação de Métodos × Herança

Métodos são invocados por despacho

Compreender o despacho na presença de herança é um aspecto sutil das linguagens OO, por exemplo

```
p : Point;
p <- new ColorPoint;
p.movePoint(1,2);
```

- O tipo estático de `p` é a classe `Point`
- Já o tipo dinâmico de `p` é a classe `ColorPoint`
- Dessa forma, a chamada `p.movePoint` deve invocar a versão da classe `ColorPoint`

Introdução à Linguagem COOL

Invocação de Métodos × Herança

Métodos são invocados por despacho

Compreender o despacho na presença de herança é um aspecto sutil das linguagens OO, por exemplo

```
p : Point;
p <- new ColorPoint;
p.movePoint(1,2);
```

- O tipo estático de `p` é a classe `Point`
- Já o tipo dinâmico de `p` é a classe `ColorPoint`
- Dessa forma, a chamada `p.movePoint` deve invocar a versão da classe `ColorPoint`

Introdução à Linguagem COOL

Invocação de Métodos × Herança

Métodos são invocados por despacho

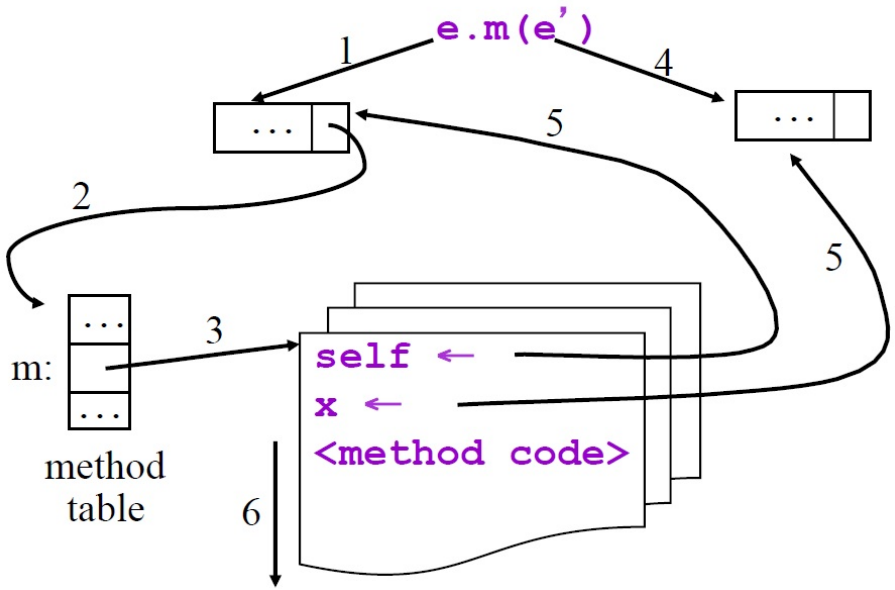
Compreender o despacho na presença de herança é um aspecto sutil das linguagens OO, por exemplo

```
p : Point;
p <- new ColorPoint;
p.movePoint(1,2);
```

- O tipo estático de `p` é a classe `Point`
- Já o tipo dinâmico de `p` é a classe `ColorPoint`
- Dessa forma, a chamada `p.movePoint` deve invocar a versão da classe `ColorPoint`

Introdução à Linguagem COOL

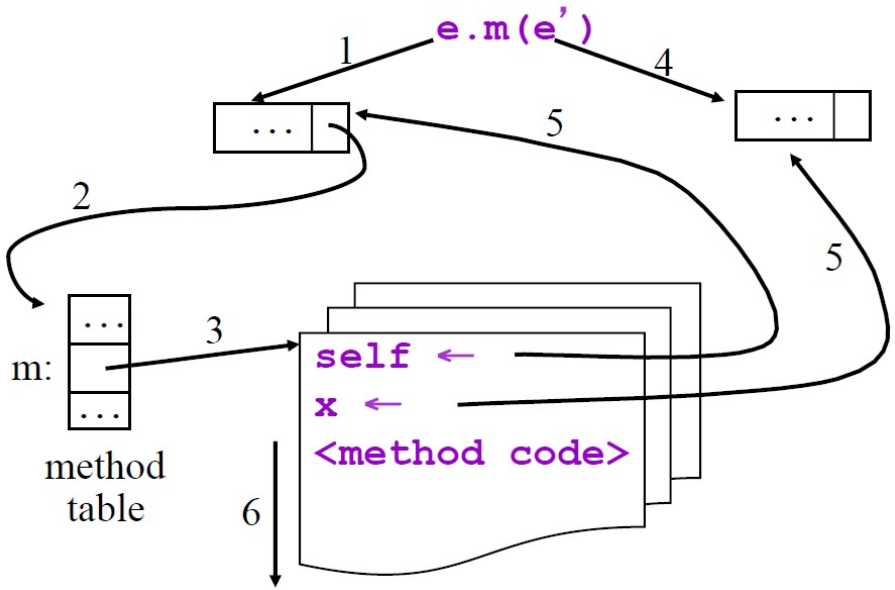
Exemplo de Invocação do Método `m` com um Parâmetro



1. Avalia `e`
2. Localiza a classe de `e`
3. Localiza o código do método `m`
4. Avalia argumento `e'`
5. Faz a "ligação" de `self` e do parâmetro `x`
6. Executa o método

Introdução à Linguagem COOL

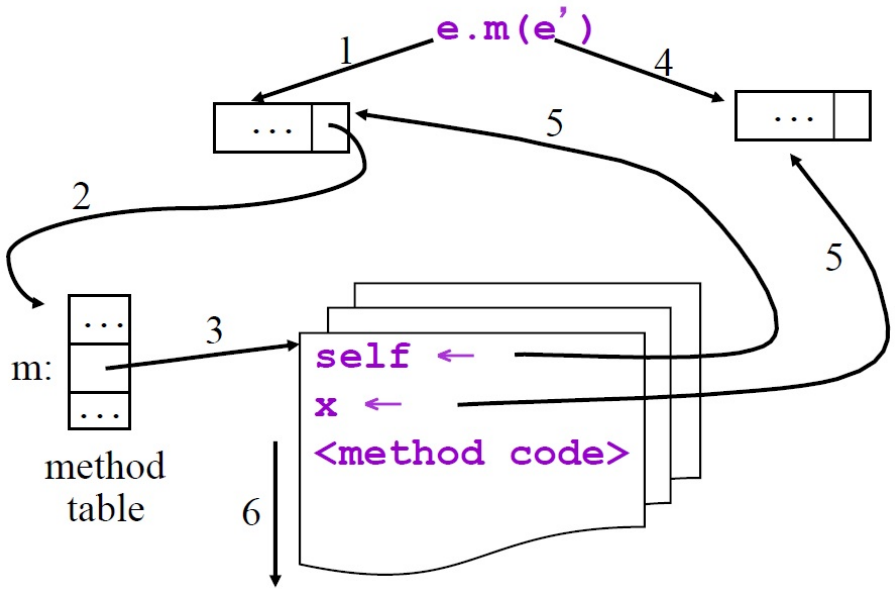
Exemplo de Invocação do Método `m` com um Parâmetro



1. Avalia `e`
2. Localiza a classe de `e`
3. Localiza o código do método `m`
4. Avalia argumento `e'`
5. Faz a "ligação" de `self` e do parâmetro `x`
6. Executa o método

Introdução à Linguagem COOL

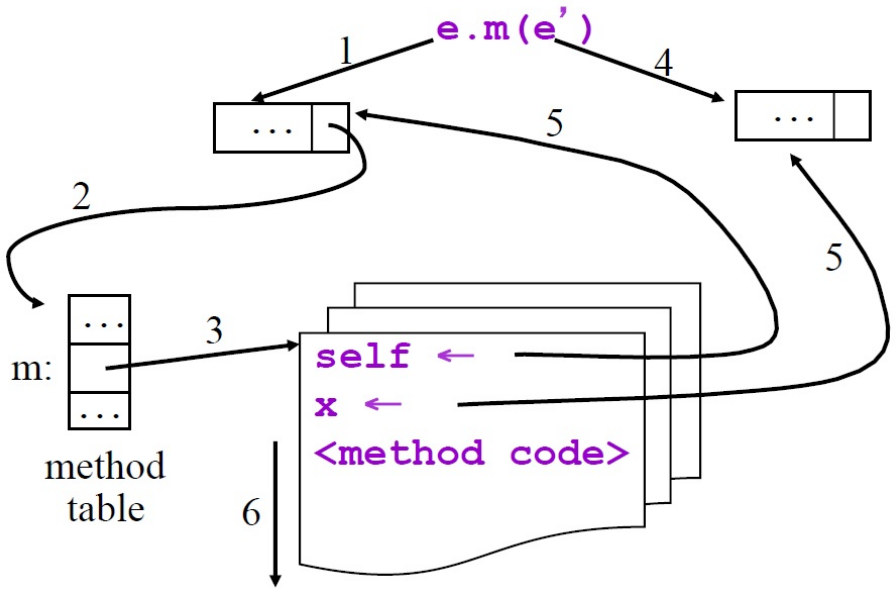
Exemplo de Invocação do Método `m` com um Parâmetro



1. Avalia `e`
2. Localiza a classe de `e`
3. Localiza o código do método `m`
4. Avalia argumento `e'`
5. Faz a "ligação" de `self` e do parâmetro `x`
6. Executa o método

Introdução à Linguagem COOL

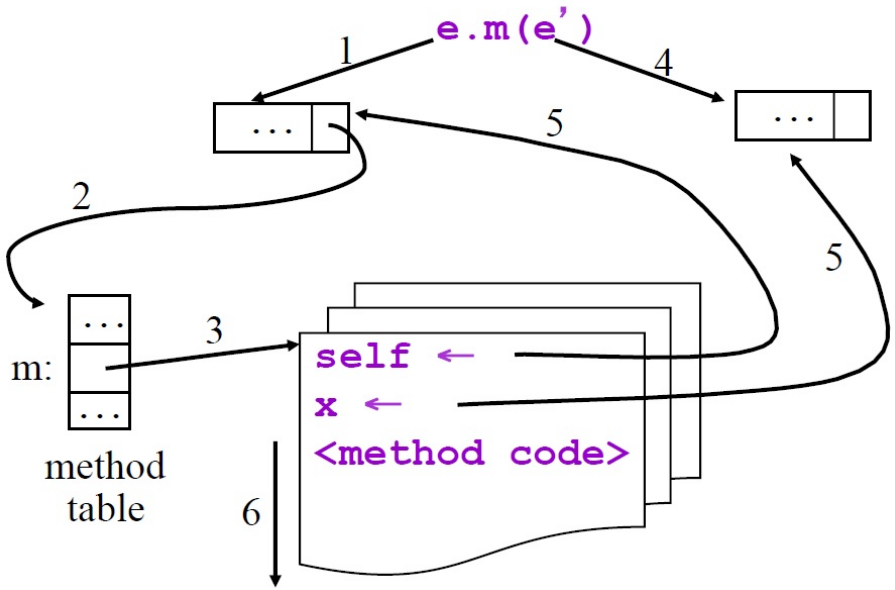
Exemplo de Invocação do Método `m` com um Parâmetro



1. Avalia `e`
2. Localiza a classe de `e`
3. Localiza o código do método `m`
4. Avalia argumento `e'`
5. Faz a "ligação" de `self` e do parâmetro `x`
6. Executa o método

Introdução à Linguagem COOL

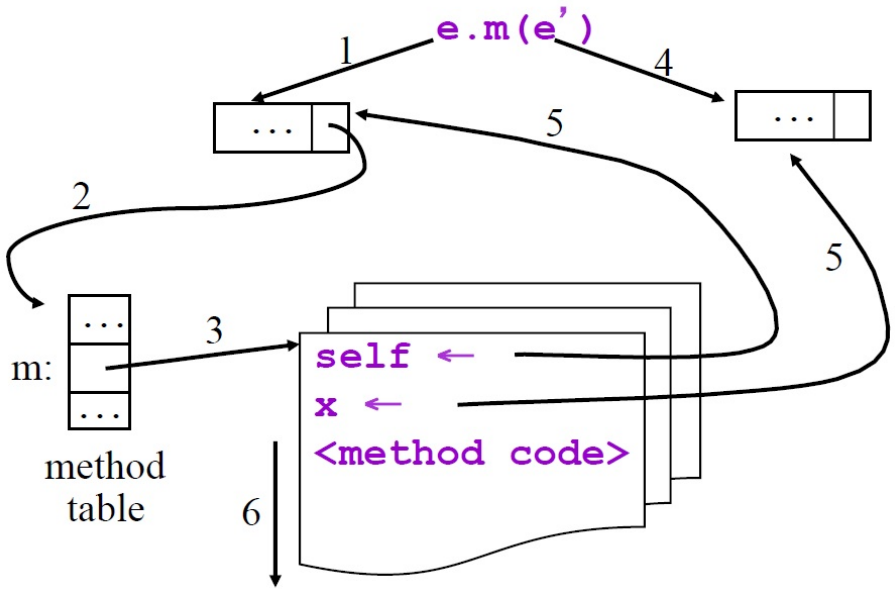
Exemplo de Invocação do Método `m` com um Parâmetro



1. Avalia `e`
2. Localiza a classe de `e`
3. Localiza o código do método `m`
4. Avalia argumento `e'`
5. Faz a "ligação" de `self` e do parâmetro `x`
6. Executa o método

Introdução à Linguagem COOL

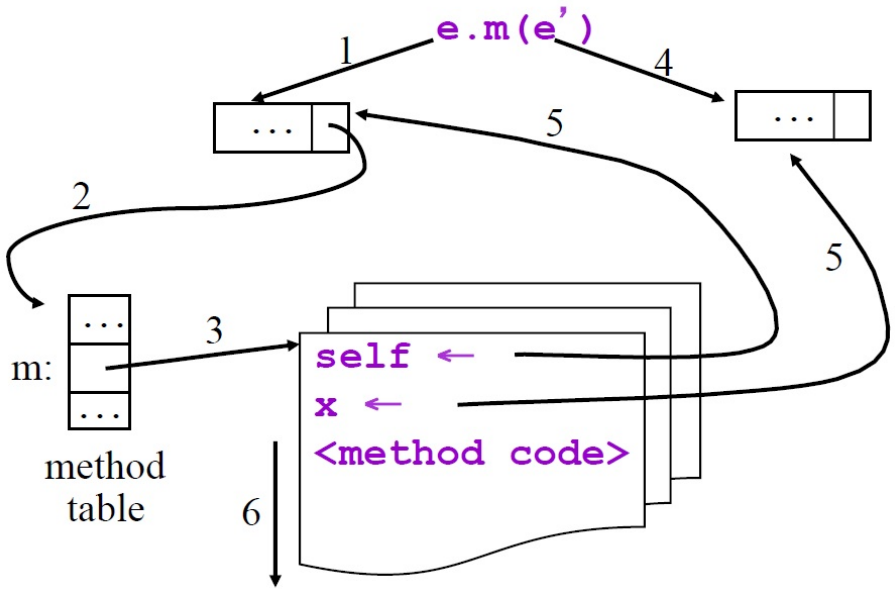
Exemplo de Invocação do Método `m` com um Parâmetro



1. Avalia `e`
2. Localiza a classe de `e`
3. Localiza o código do método `m`
4. Avalia argumento `e'`
5. Faz a "ligação" de `self` e do parâmetro `x`
6. Executa o método

Introdução à Linguagem COOL

Exemplo de Invocação do Método `m` com um Parâmetro



1. Avalia `e`
2. Localiza a classe de `e`
3. Localiza o código do método `m`
4. Avalia argumento `e'`
5. Faz a "ligação" de `self` e do parâmetro `x`
6. Executa o método

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de “expressões”

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando “*Case*”: `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Expressões

Linguagem COOL ⇒ linguagem de "expressões"

- Toda expressão possui um tipo e um valor
- Expressões aritméticas e lógicas
- Atribuição: `x <- E`
- Repetição: `while E loop E pool`
- Alternativa: `if E then E else E fi`
- Comando "*Case*": `case E of x : Type => E; ... esac`
- Primitivas de E/S: `out_string(s)`, `in_string()`, ...

Características ausentes

- vetores, operações de ponto flutuante, exceções, ...

Introdução à Linguagem COOL

Gerenciamento de Memória em COOL

A alocação de memória é realizada cada vez que `new` é invocado

A desalocação de memória é realizada de forma automática quando um objeto não é mais "alcançável"

A liberação de memória fica a cargo de um coletor de lixo (GC – *Garbage Collector*)

- Portanto, um GC que deverá existir em uma implementação de COOL

Introdução à Linguagem COOL

Gerenciamento de Memória em COOL

A alocação de memória é realizada cada vez que `new` é invocado

A desalocação de memória é realizada de forma automática quando um objeto não é mais "alcançável"

A liberação de memória fica a cargo de um coletor de lixo (GC – *Garbage Collector*)

- Portanto, um GC que deverá existir em uma implementação de COOL

Introdução à Linguagem COOL

Gerenciamento de Memória em COOL

A alocação de memória é realizada cada vez que `new` é invocado

A desalocação de memória é realizada de forma automática quando um objeto não é mais "alcançável"

A liberação de memória fica a cargo de um coletor de lixo (GC – *Garbage Collector*)

- Portanto, um GC que deverá existir em uma implementação de COOL

Introdução à Linguagem COOL

Gerenciamento de Memória em COOL

A alocação de memória é realizada cada vez que `new` é invocado

A desalocação de memória é realizada de forma automática quando um objeto não é mais "alcançável"

A liberação de memória fica a cargo de um coletor de lixo (GC – *Garbage Collector*)

- Portanto, um GC que deverá existir em uma implementação de COOL

Programas na Linguagem COOL

Exemplo N.01

```
class Main {
    main(): Int { 1 };
};
```

Exemplo N.02

```
class Main {
    i : IO <- new IO;
    main(): Int {
        {
            i.out_string("Hello World!\n");
            1;
        }
    };
};
```

Programas na Linguagem COOL

Exemplo N.01

```
class Main {
    main(): Int { 1 };
};
```

Exemplo N.02

```
class Main {
    i : IO <- new IO;
    main(): Int {
        {
            i.out_string("Hello World!\n");
            1;
        }
    };
};
```

Programas na Linguagem COOL

Exemplo N.03

```
class Main {
    i :  IO <- new IO;
    main():  IO { i.out_string("Hello World!\n") };
};
```

Exemplo N.04

```
class Main {
    i :  IO <- new IO;
    main():  Object { i.out_string("Hello World!\n") };
};
```

Programas na Linguagem COOL

Exemplo N.03

```
class Main {
    i :  IO <- new IO;
    main():  IO { i.out_string("Hello World!\n") };
};
```

Exemplo N.04

```
class Main {
    i :  IO <- new IO;
    main():  Object { i.out_string("Hello World!\n") };
};
```

Programas na Linguagem COOL

Exemplo N.05

```
class Main {
    main():  Object { (new IO).out_string("Hello World!\n") };
};
```

Exemplo N.06

```
class Main inherits IO {
    main():  Object { self.out_string("Hello World!\n") };
};
```

Exemplo N.07

```
class Main inherits IO {
    main():  Object { out_string("Hello World!\n") };
};
```

Programas na Linguagem COOL

Exemplo N.05

```
class Main {
    main():  Object { (new IO).out_string("Hello World!\n") };
};
```

Exemplo N.06

```
class Main inherits IO {
    main():  Object { self.out_string("Hello World!\n") };
};
```

Exemplo N.07

```
class Main inherits IO {
    main():  Object { out_string("Hello World!\n") };
};
```

Programas na Linguagem COOL

Exemplo N.05

```
class Main {
    main(): Object { (new IO).out_string("Hello World!\n") };
};
```

Exemplo N.06

```
class Main inherits IO {
    main(): Object { self.out_string("Hello World!\n") };
};
```

Exemplo N.07

```
class Main inherits IO {
    main(): Object { out_string("Hello World!\n") };
};
```

Programas na Linguagem COOL

Exemplo N.08

```
class Main {
    main():  Object {
        (new IO).out_string((new IO).in_string().concat("\n"))
    };
};
```

Exemplo N.09

```
class Main inherits A2I {
    main():  Object {
        (new IO).out_string(
            i2a(
                a2i((new IO).in_string())+1
            ).concat("\n")
        )
    };
};
```

Programas na Linguagem COOL

Exemplo N.08

```
class Main {
    main():  Object {
        (new IO).out_string((new IO).in_string().concat("\n"))
    };
};
```

Exemplo N.09

```
class Main inherits A2I {
    main():  Object {
        (new IO).out_string(
            i2a(
                a2i((new IO).in_string())+1
            ).concat("\n")
        )
    };
};
```

Programas na Linguagem COOL

Exemplo N.10

```
class Main inherits A2I {
    main(): Object {
        (new IO).out_string(
            i2a(
                fact(a2i((new IO).in_string()))
            ).concat("\n")
        )
    };
    fact(i : Int): Int {
        if (i = 0) then
            1
        else
            i * fact(i-1)
        fi
    };
};
```

Programas na Linguagem COOL

Exemplo N.11

```
class Main inherits A2I {
    main(): Object {
        (new IO).out_string(
            i2a(fact(a2i((new IO).in_string()))).concat("\n")
        )
    };
    fact(i : Int): Int {
        let fact: Int <- 1 in {
            while (not (i = 0)) loop
                {
                    fact <- fact * i;
                    i <- i - 1;
                }
            pool;
            fact;
        }
    };
};
```

Programas na Linguagem COOL

Exemplo N.12

```
class Main inherits IO {
    main():  Object {
        let hello:  String <- "Hello ",
            world:  String <- "World!",
            newline:  String <- "\n"
        in
            out_string(hello.concat(world.concat(newline)))
    };
};
```

Programas na Linguagem COOL

Exemplo N.13

```
class List {
    item:  String;
    next:  List;
    init(i:  String, n:  List):  List {
        item <- i;
        next <- n;
        self;
    };
    flatten():  String {
        if (isvoid next) then
            item
        else
            item.concat(next.flatten())
        fi
    };
};
```

Programas na Linguagem COOL

Exemplo N.13 (cont.)

```
class Main inherits IO {
   main():  Object {
      let hello:  String <- "Hello ",
          world:  String <- "World!",
          newline:  String <- "\n",
          nil:  List,
          list:  List <-
             (new List).init(hello,
                (new List).init(world,
                   (new List).init(newline,nil)))
      in
          out_string(list.flatten())
   };
};
```

Programas na Linguagem COOL

Exemplo N.14

```
class List inherits A2I {
    item:  Object;
    next:  List;

    init(i:  Object, n:  List):  List {
        item <- i; next <- n; self;
    };

    flatten():  String {
        let string:  String <-
            case item of
                i:  Int => i2a(i);
                s:  String => s;
                o:  Object => { abort(); ""; };
            esac
        in
            if (isvoid next) then string
            else string.concat(next.flatten())fi
    };
};
```

Programas na Linguagem COOL

Exemplo N.14 (cont.)

```
class Main inherits IO {
   main():  Object {
      let hello:  String <- "Hello ",
          world:  String <- "World!",
          i:  Int <- 42,
          newline:  String <- "\n",
          nil:  List,
          list:  List <-
             (new List).init(hello,
                (new List).init(world,
                   (new List).init(i,
                      (new List).init(newline,nil))))
      in
          out_string(list.flatten())
   };
};
```

Introdução à Linguagem COOL

Projeto de Curso

O projeto consiste em um compilador completo

- COOL $\Longrightarrow$ Linguagem *assembly* do MIPS
- Sem otimização (Apenas como tarefa extra!)

Implementação dividida em 04 partes

Tempo do semestre é adequado para realização da implementação

- Porém inicie o **mais breve possível** e siga todas as instruções

Implementações podem ser feitas em equipes

- Máximo de 04 alunos

Introdução à Linguagem COOL

Projeto de Curso

O projeto consiste em um compilador completo

- COOL $\Longrightarrow$ Linguagem *assembly* do MIPS
- Sem otimização (Apenas como tarefa extra!)

Implementação dividida em 04 partes

Tempo do semestre é adequado para realização da implementação

- Porém inicie o **mais breve possível** e siga todas as instruções

Implementações podem ser feitas em equipes

- Máximo de 04 alunos

Introdução à Linguagem COOL

Projeto de Curso

O projeto consiste em um compilador completo

- COOL $\Longrightarrow$ Linguagem *assembly* do MIPS
- Sem otimização (Apenas como tarefa extra!)

Implementação dividida em 04 partes

Tempo do semestre é adequado para realização da implementação

- Porém inicie o **mais breve possível** e siga todas as instruções

Implementações podem ser feitas em equipes

- Máximo de 04 alunos

Introdução à Linguagem COOL

Projeto de Curso

O projeto consiste em um compilador completo

- COOL $\Longrightarrow$ Linguagem *assembly* do MIPS
- Sem otimização (Apenas como tarefa extra!)

Implementação dividida em 04 partes

Tempo do semestre é adequado para realização da implementação

- Porém inicie o **mais breve possível** e siga todas as instruções

Implementações podem ser feitas em equipes

- Máximo de 04 alunos

Introdução à Linguagem COOL

Projeto de Curso

O projeto consiste em um compilador completo

- COOL $\Longrightarrow$ Linguagem *assembly* do MIPS
- Sem otimização (Apenas como tarefa extra!)

Implementação dividida em 04 partes

Tempo do semestre é adequado para realização da implementação

- Porém inicie o **mais breve possível** e siga todas as instruções

Implementações podem ser feitas em equipes

- Máximo de 04 alunos

Introdução à Linguagem COOL

Projeto de Curso

O projeto consiste em um compilador completo

- COOL $\Longrightarrow$ Linguagem *assembly* do MIPS
- Sem otimização (Apenas como tarefa extra!)

Implementação dividida em 04 partes

Tempo do semestre é adequado para realização da implementação

- Porém inicie o **mais breve possível** e siga todas as instruções

Implementações podem ser feitas em equipes

- Máximo de 04 alunos