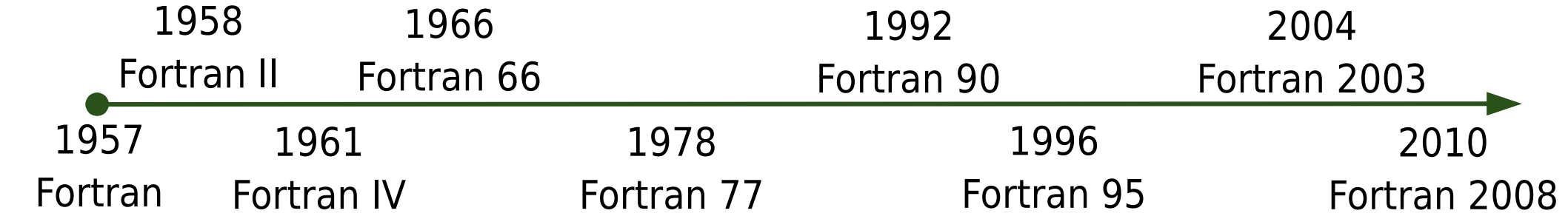
85ª SEQ/UFRJ
20 a 24 de agosto/2018

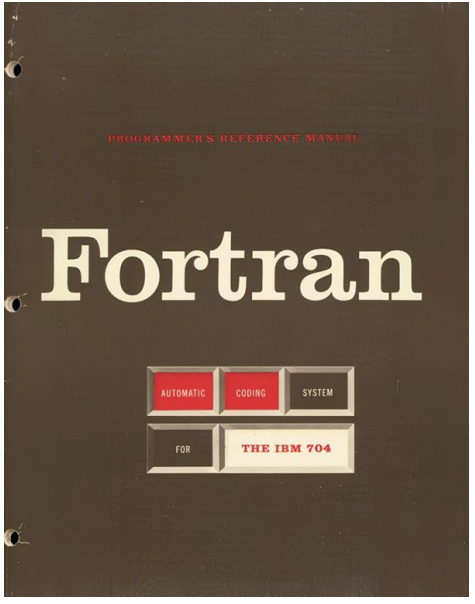
Minicurso

Programação básica em Fortran

Rafael Pereira do Carmo
email: rcarmo@peq.coppe.ufrj.br

Fortran – **For**mula **Tran**slation



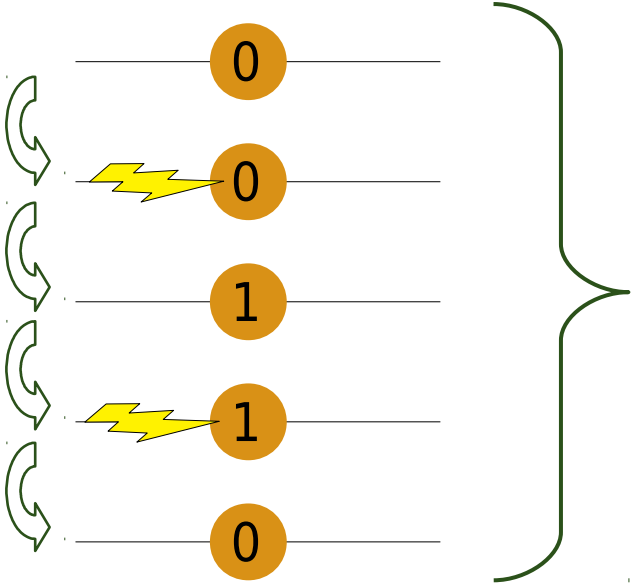


Por que usar Fortran?

- Mais simples que C e C++;
- Mais rápido que Matlab e Python;
- Muitos programas escritos em Fortran (legado).

Como funciona o computador?

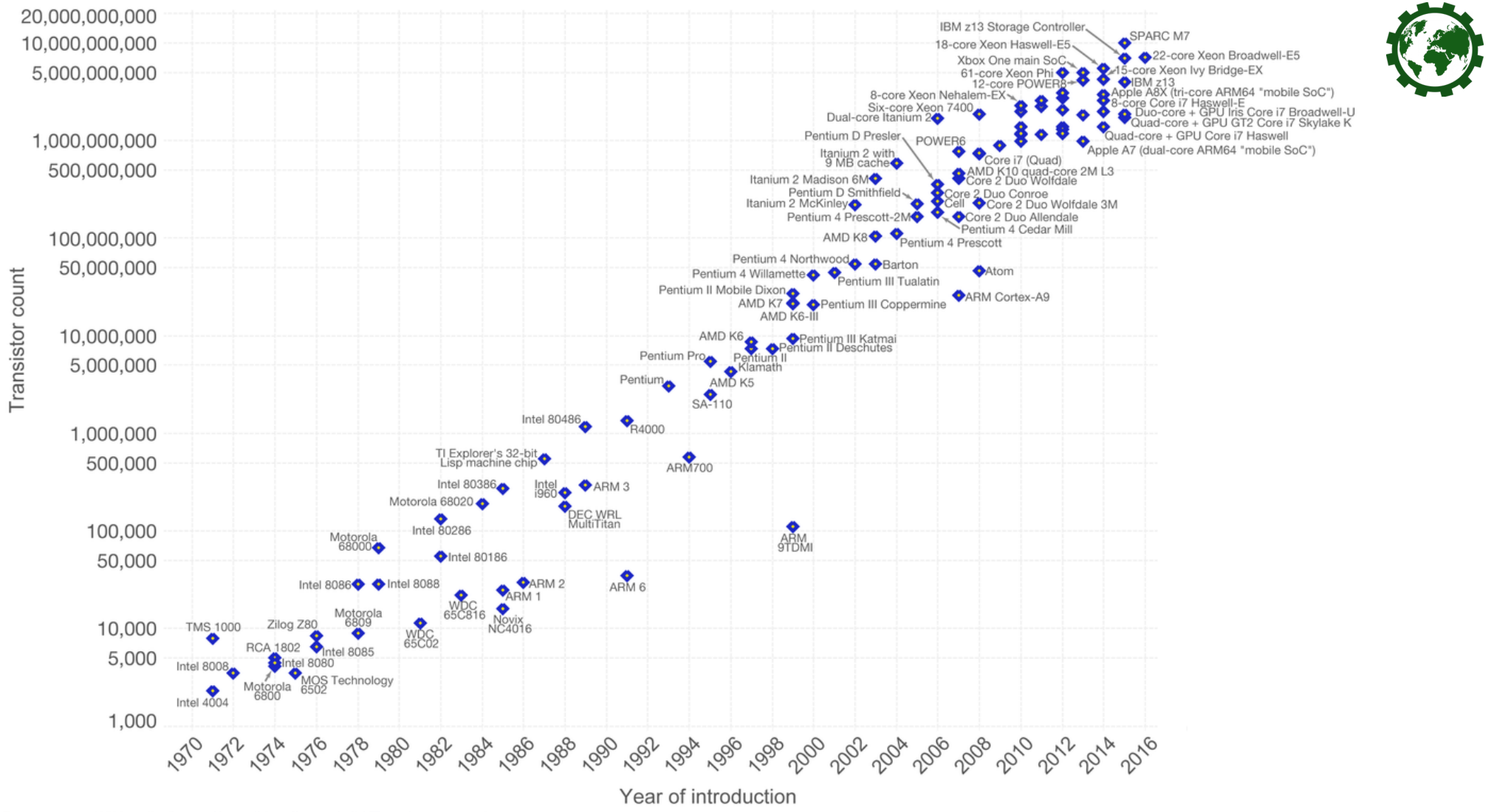
Transistor



Bit (Binary Digit):
unidade de informação

0 1 0 1 0 0 1 1

8 Bits = 1 Byte (Binary Term)

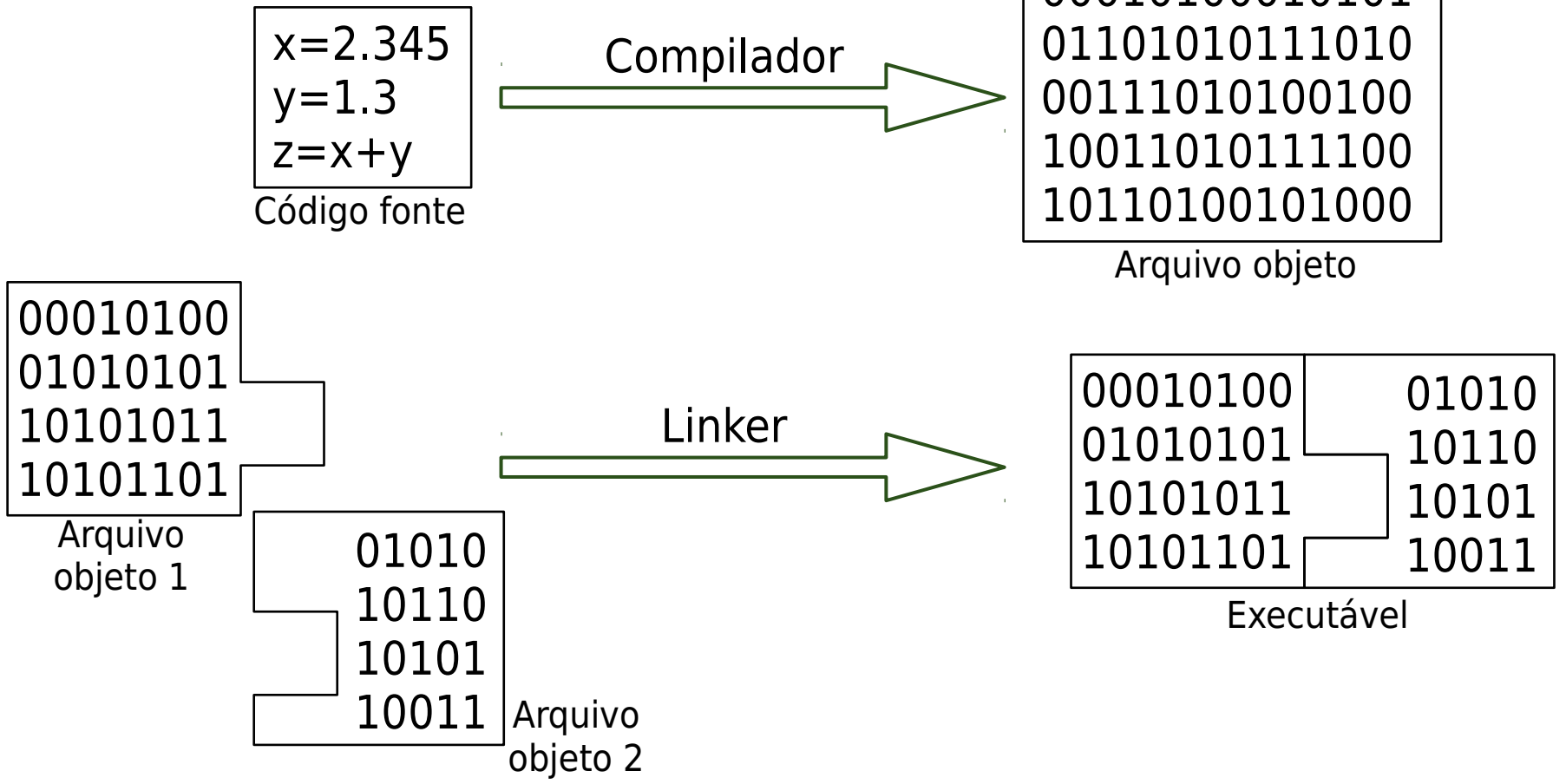


Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Como funciona o computador?

Curiosidade: Os menores transistores de um microprocessador comercial, hoje, possuem cerca de 14 nm. Abaixo de 7 nm, os átomos de silício encontram-se tão próximos que os elétrons sofrem tunelamento quântico. Assim, é preciso explorar a utilização de novos materiais para confecção dos transistores (ou partir para computação quântica).

Como é gerado um programa?



Como é gerado um programa?

Observação 1: Na prática, um arquivo objeto ou um executável não são compostos apenas por números binários. Os endereços de memória costumam ser representados por meio do sistema hexadecimal.

```
00000012: 00111110 00000000 00000001 00000000 00000000 00000000  >.....
00000018: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000001e: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000024: 00000000 00000000 00000000 00000000 10000000 10000011  ......
0000002a: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000030: 00000000 00000000 00000000 00000000 01000000 00000000  ....@.
00000036: 00000000 00000000 00000000 00000000 01000000 00000000  ....@.
0000003c: 00001101 00000000 00001100 00000000 01010101 01001000  ....UH
00000042: 10001001 11100101 01000001 01010100 01010011 01001000  ..ATSH
00000048: 10000001 11101100 11000000 01100001 00000000 00000000  ...a..
0000004e: 01001000 11000111 00000101 00000000 00000000 00000000  H.....
00000054: 00000000 00000000 00000000 00000000 00000000 01001000  .....H
0000005a: 11000111 00000101 00000000 00000000 00000000 00000000  ......
00000060: 00000000 00000000 00000000 00000000 01001000 11000111  ....H.
00000066: 00000101 00000000 00000000 00000000 00000000 00000000  ......
0000006c: 00000000 00000000 00000000 01001000 11000111 00000101  ...H..
00000072: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000078: 00000000 00000000 01001000 11000111 00000101 00000000  ..H...
0000007e: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000084: 00000000 01001000 11000111 00000101 00000000 00000000  .H....
0000008a: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000090: 01001000 10001101 00000101 00000000 00000000 00000000  H.....
00000096: 00000000 01001000 10001001 10000101 01111000 10011110  .H..x.
```

Como é gerado um programa?

Observação 2: Não confundir linguagem de máquina com Assembly!

```
0:    55                   push   %rbp
1:    48 89 e5             mov    %rsp,%rbp
4:    41 54                push   %r12
6:    53                   push   %rbx
7:    48 81 ec c0 61 00 00 sub    $0x61c0,%rsp
e:    48 c7 05 00 00 00 00 movq   $0x0,0x0(%rip)        # 19 <MAIN__+0x19>
15:   00 00 00 00
19:   48 c7 05 00 00 00 00 movq   $0x0,0x0(%rip)        # 24 <MAIN__+0x24>
20:   00 00 00 00
24:   48 c7 05 00 00 00 00 movq   $0x0,0x0(%rip)        # 2f <MAIN__+0x2f>
2b:   00 00 00 00
2f:   48 c7 05 00 00 00 00 movq   $0x0,0x0(%rip)        # 3a <MAIN__+0x3a>
36:   00 00 00 00
3a:   48 c7 05 00 00 00 00 movq   $0x0,0x0(%rip)        # 45 <MAIN__+0x45>
41:   00 00 00 00
45:   48 c7 05 00 00 00 00 movq   $0x0,0x0(%rip)        # 50 <MAIN__+0x50>
4c:   00 00 00 00
50:   48 8d 05 00 00 00 00 lea    0x0(%rip),%rax        # 57 <MAIN__+0x57>
57:   48 89 85 78 9e ff ff mov    %rax,-0x6188(%rbp)
5e:   c7 85 80 9e ff ff 5c movl   $0x5c,-0x6180(%rbp)
65:   00 00 00
68:   48 8d 05 00 00 00 00 lea    0x0(%rip),%rax        # 6f <MAIN__+0x6f>
6f:   48 89 85 a0 9e ff ff mov    %rax,-0x6160(%rbp)
76:   c7 85 9c 9e ff ff 0e movl   $0xe,-0x6164(%rbp)
```

Como é gerado um programa?

Linguagem de baixo nível

Assembly

Baixo nível de abstração: assemellha-se mais à linguagem de máquina

Linguagem de alto nível

Fortran **C**

C++ **Python**

Alto nível de abstração: assemellha-se mais à linguagem humana

Compiladores

Mais usados

gfortran

Intel® Fortran

De graça

Obs: LD (linker da GNU)

Licença acadêmica

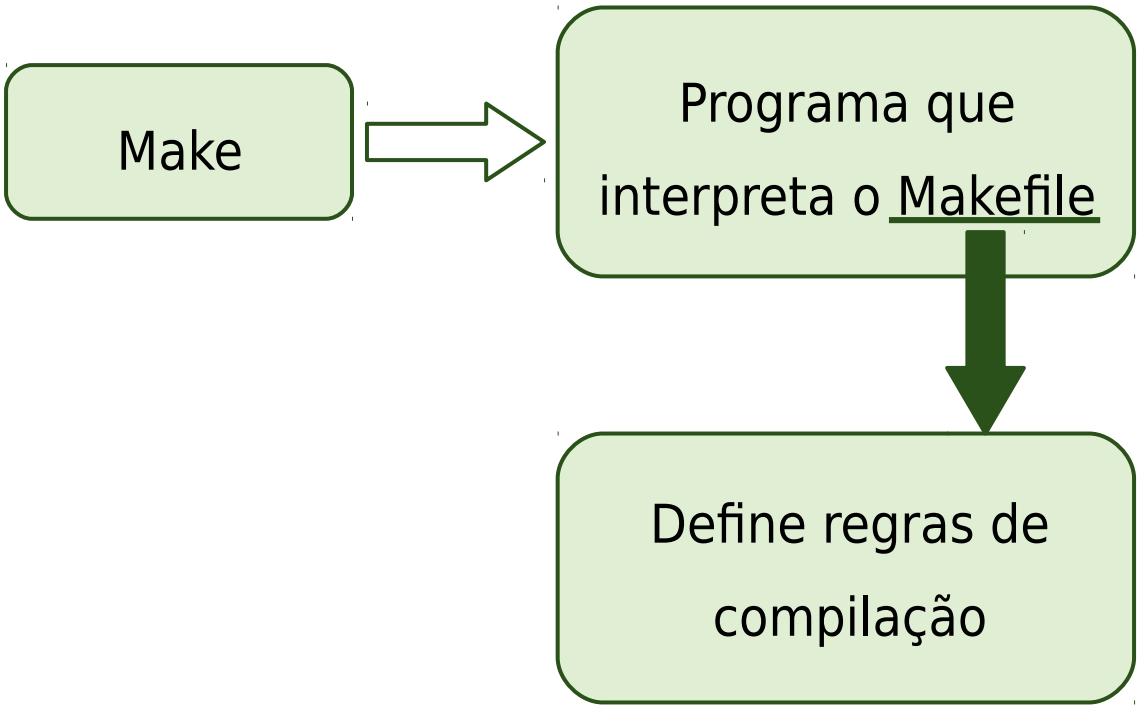
Outros

IBM XL Fortran

absoft Pro Fortran

SilverFrost

Make

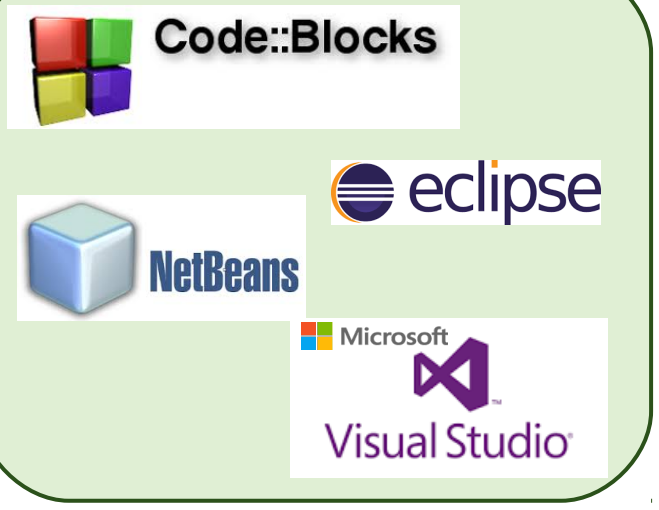

Make
Programa que interpreta o Makefile
Define regras de compilação

IDE

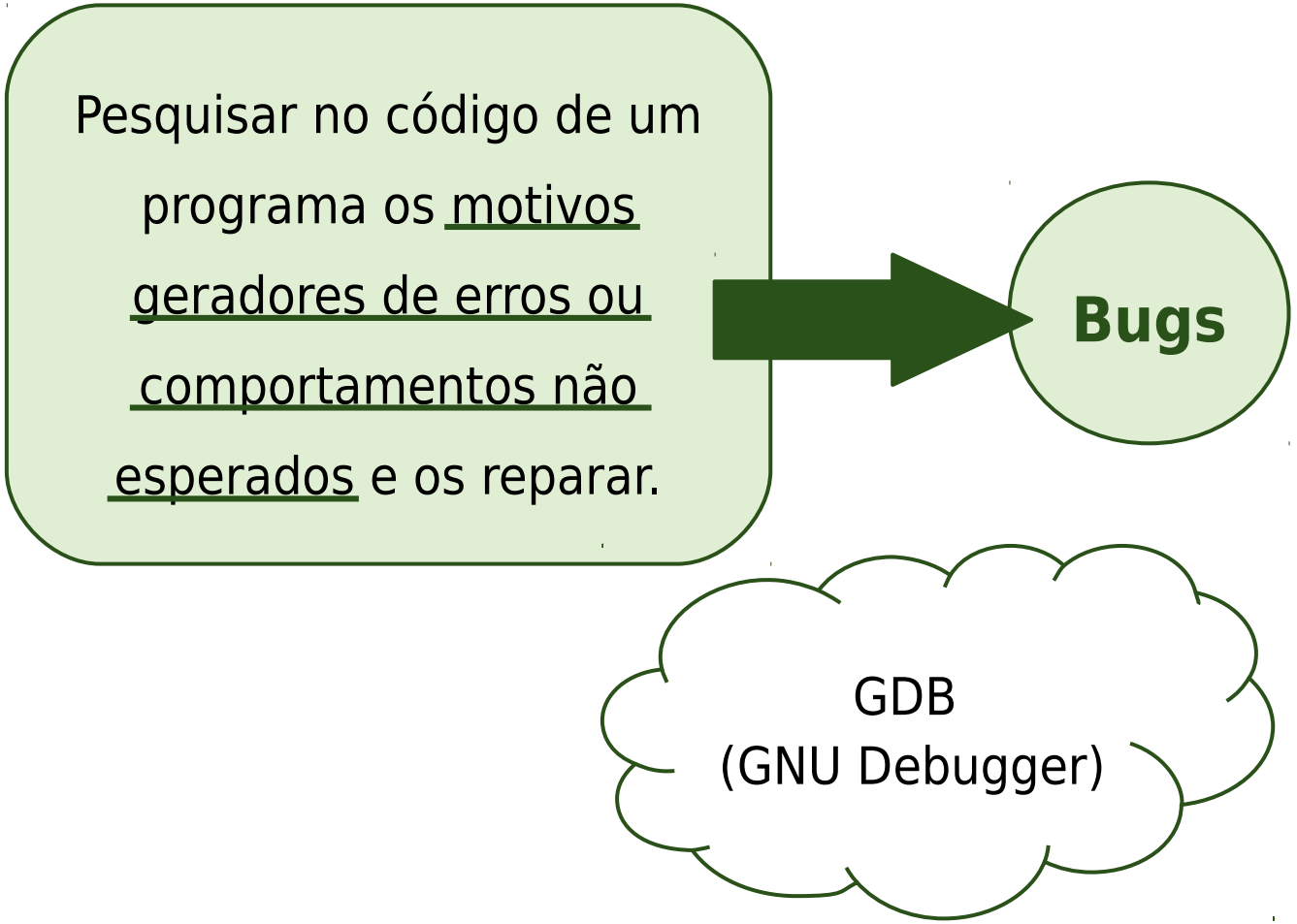
Integrated **D**evelopment **E**nvironment

(Ambiente de Desenvolvimento Integrado)

- Editor de texto
- Compilador
- Linker
- Makefile (Receita)
- Execução do programa
- Debugger



Debugger


Pesquisar no código de um programa os motivos geradores de erros ou comportamentos não esperados e os reparar.
Bugs
GDB
(GNU Debugger)

Criando um projeto no Code::Blocks

- Abra a pasta “codeblocks-17.12-nosetup”;
- Execute o CODE::BLOCKS clicando duas vezes no arquivo codeblocks.exe;
- No programa, clique em file>new>project>empty project
- Quando uma janela abrir, clique em next e, em seguida, insira o nome e local do projeto. Clique em next.
- Na área “Compiler”, selecione “GNU Fortran Compiler”. Clique em “finish”.
- Para adicionar um novo arquivo, clique no ícone “New file” no canto superior esquerdo.
- Selecione empty file.
- No tipo do arquivo, selecione “fortran files”. Insira o nome do arquivo e clique em Ok.

Meu primeiro programa em Fortran

1) Para começar a escrever um programa, escreva “**program** nome_do_programa” e, abaixo, “**end program**” para determinar o fim do programa.

2) Para fazer comentários ao longo do programa, digite “!” logo antes do que deseja escrever.

3) Escreva a seguinte linha de comando: print*, “Meu primeiro programa!”

```
1
2 !Meu primeiro programa (isso aqui é um comentário)
3 program meu_programa !Começo de um programa
4
5     print*, "Meu primeiro programa!"
6
7 end program !Fim de um programa
```

4) **Code::Blocks**: Clique em  no canto esquerdo superior para compilar o programa e, em seguida, clique em  para rodar o programa.

Meu primeiro programa em Fortran

Obs 1: Faça **indentação** do programa, ou seja, dê alguns espaços no início das linhas para facilitar a identificação de diferentes blocos de código.

Obs 2: A linguagem Fortran **não** é ***case sensitive***, ou seja, não diferencia letras maiúsculas e minúsculas.

```
integer variavel
integer VaRiAvEl
```

```
     integer VaRiAvEl
                     1
Error: Symbol ‘variavel’ at (1) already has basic type of INTEGER
```

Declaração de variáveis

Tipos de variáveis

- integer
- real
- logical
- character
- complex

As variáveis devem ser declaradas no início do programa

```
program meu_programa

    implicit none
    !Bloco de declarações de variáveis
    integer meu_int
    real meu_real
    logical meu_log
    character meu_char
    complex meu_complex

    !Resto do programa embaixo
```

Sempre escrever “implicit none” no início do programa!!11!1!

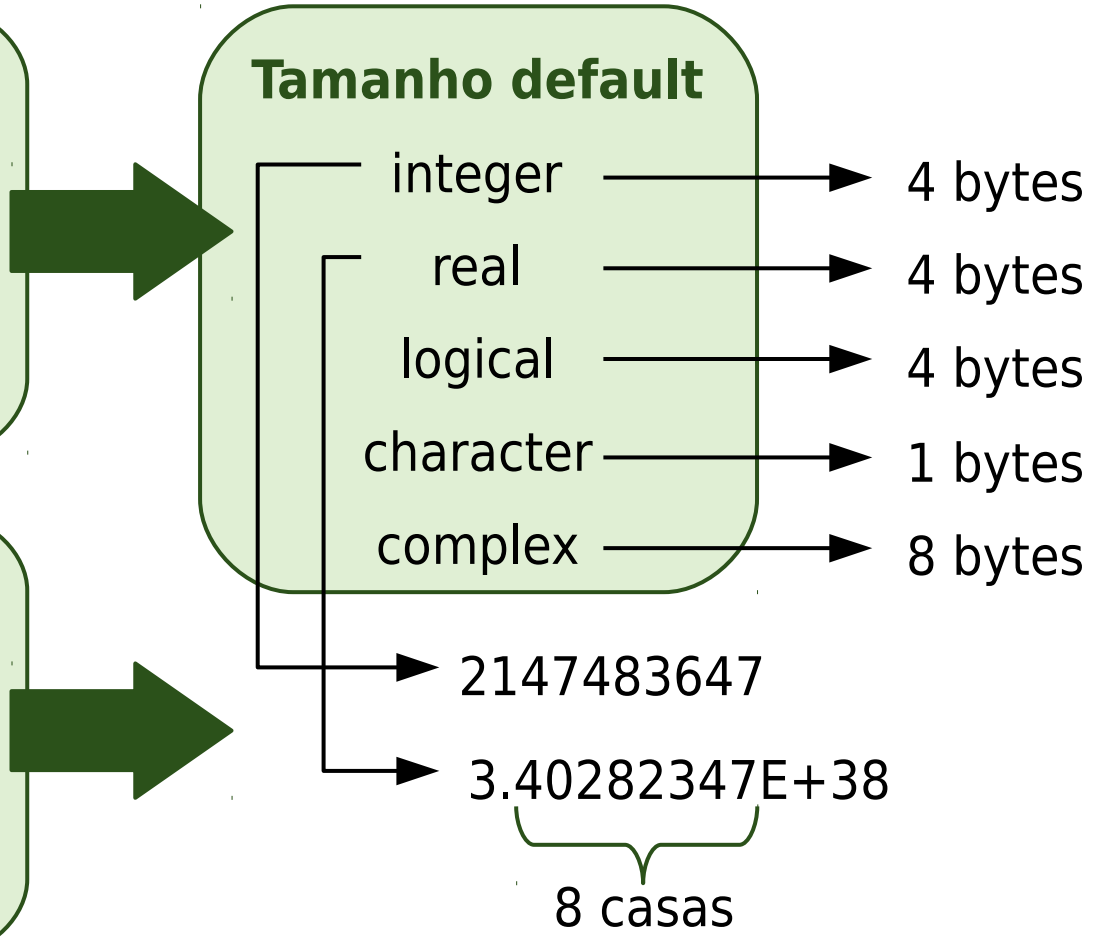
Declaração de variáveis

Combine as funções “print” e “sizeof()” para descobrir o tamanho default de cada tipo de variável.

```
print*, "int: ", sizeof(meu_int), "bytes"
```

Agora, combine as funções “print” e “huge()” para descobrir o maior valor que variáveis “real” e “integer” podem ter.

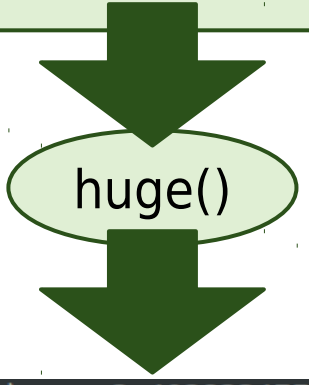
```
print*, "maior int: ", huge(meu_int)
```



Declaração de variáveis

Obs: É posível alterar o espaço que cada tipo de variável ocupa na memória:

```
integer(2) meu_int_p !inteiro de 2 bytes
integer(4) meu_int !Default: inteiro de 4 bytes
integer(8) meu_int_g !inteiro de 8 bytes
real(4) meu_real !Default: real de 4 bytes
real(8) meu_real_d !real de 8 bytes (dupla precisão)
```



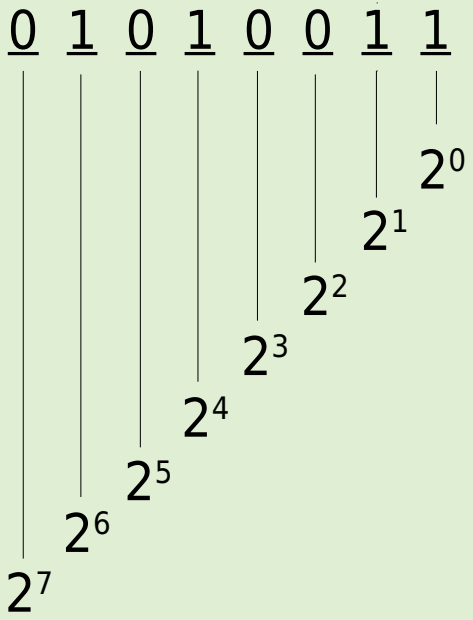

```
maior real(4):    3.40282347E+38
maior real(8):    1.7976931348623157E+308
maior int(2):   32767
maior int(4):   2147483647
maior int(8):   9223372036854775807
```

16 casas decimais

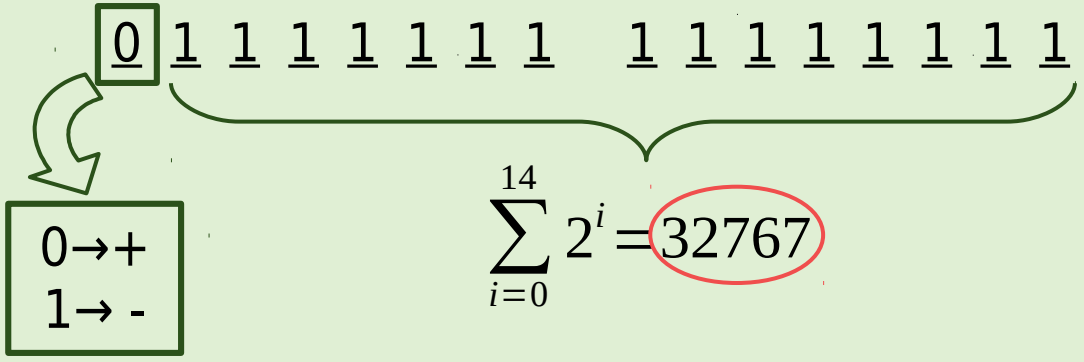
Declaração de variáveis

Curiosidade: Qual a relação entre o espaço ocupado e os valores de um integer?

8 Bits = 1 Byte (Binary Term)



Integer(2) – 2 bytes



0→+
1→ -

$$\sum_{i=0}^{14} 2^i = 32767$$

Obs:

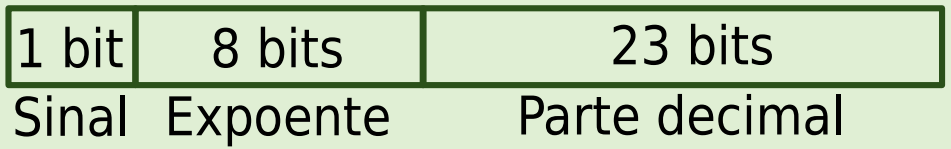
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

= -32768

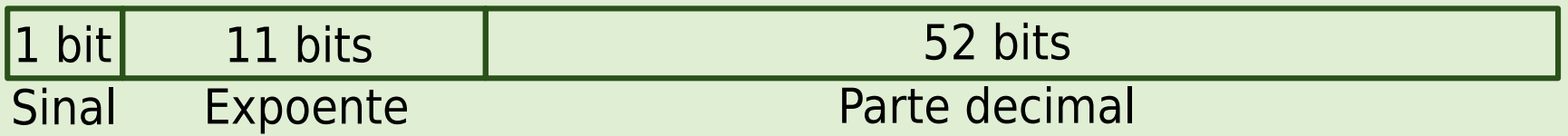
Declaração de variáveis

Curiosidade: Qual a relação entre o espaço ocupado e os valores de um real?

Real(4) - 4 bytes = 32 bits



Real(8) - 8 bytes



Declaração de variáveis

Obs 1: Posso declarar quantas variáveis eu quiser na mesma linha.

```
integer a, b, c, d, e, f
```

Obs 3: Posso declarar uma constante (não poderá ser modificada ao longo do programa)

```
real(8), parameter :: pi = 3.1415d0
```

Obs 2: Posso definir o valor das variáveis ao mesmo tempo que as declaro.

```
real(8) :: r1=1.234
real(8) :: r2=1.234d0, r3=1.234d-2, r4=1.234d+2, r5
```

print

```
r1 =   1.2339999675750732
r2 =   1.2340000000000000
r3 =   1.2340000000000000E-002
r4 =   123.40000000000001
```

Para definir valores durante a declaração, é preciso colocar "::" após o tipo

Declaração de variáveis

Array: Variável que armazena sequencialmente diversos valores.

```
real(8) :: array1(3)
real(8) :: array2(3)=0.d0
real(8) :: array3(3)=1.2d0
real(8) :: array4(3)=[1.2d0,2.3d0,3.4d0]
```


```
array1 =   0.0000000000000000       -6.6647744572169283E-141   0.0000000000000000
array2 =   0.0000000000000000        0.0000000000000000        0.0000000000000000
array3 =   1.2000000000000000        1.2000000000000000        1.2000000000000000
array4 =   1.2000000000000000        2.2999999999999998        3.3999999999999999
```

Declaração de variáveis

```
real(8) :: array1_2D(2,3)=transpose(reshape([1.1d0,1.2d0,1.3d0,2.1d0,2.2d0,2.3d0],[3,2]))
real(8) :: array2_2D(2,2)=1.d0

array2_2D(1,2) = 2.0d0
array2_2D(2,1) = 3.0d0
array2_2D(2,2) = 4.0d0
```

print

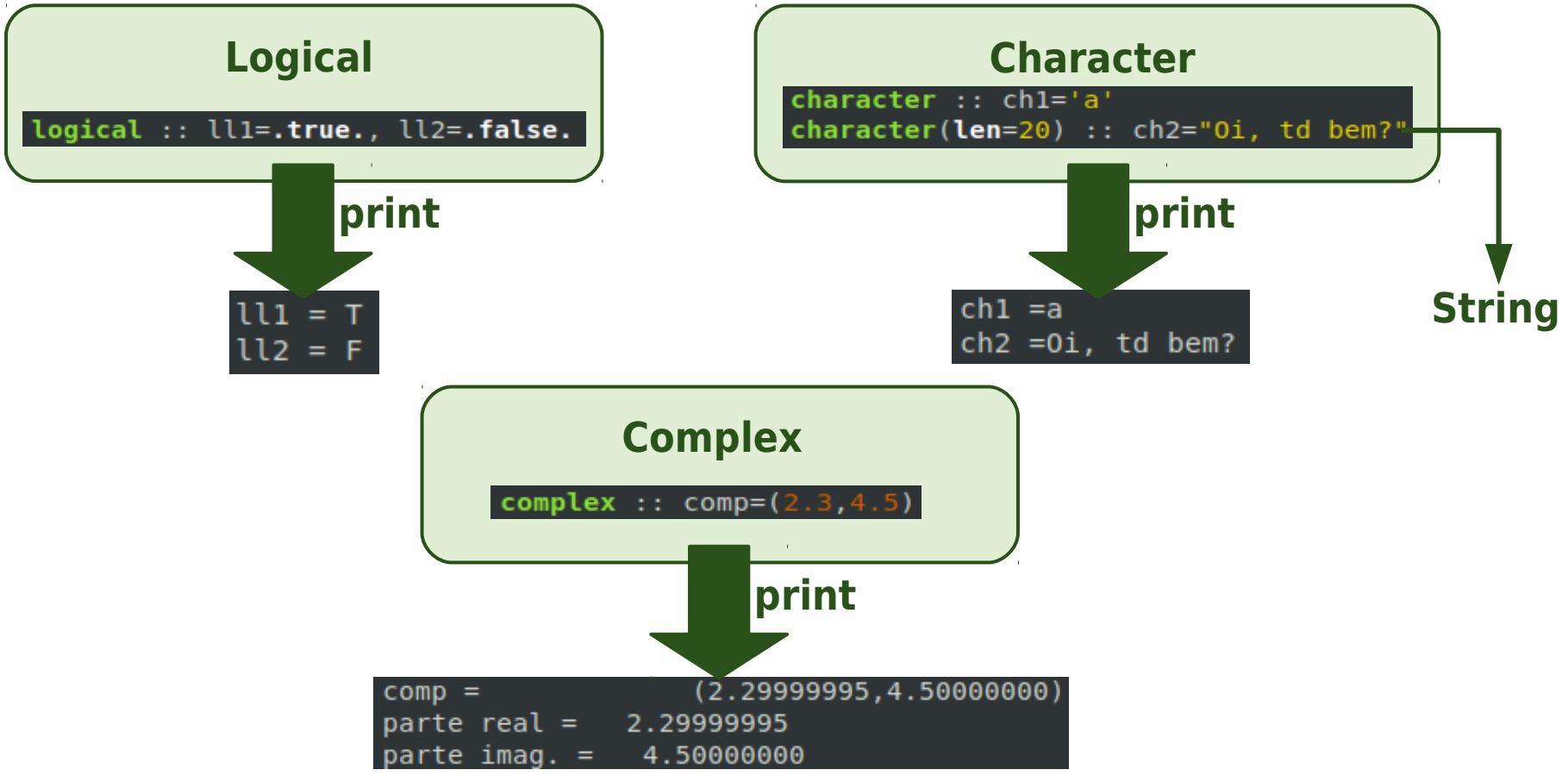
```
array2_2D =   1.0000000000000000        3.0000000000000000        2.0000000000000000        4.0000000000000000
```

Como lemos ← **Cuidado** → **Como o Fortran armazena na memória**

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Declaração de variáveis



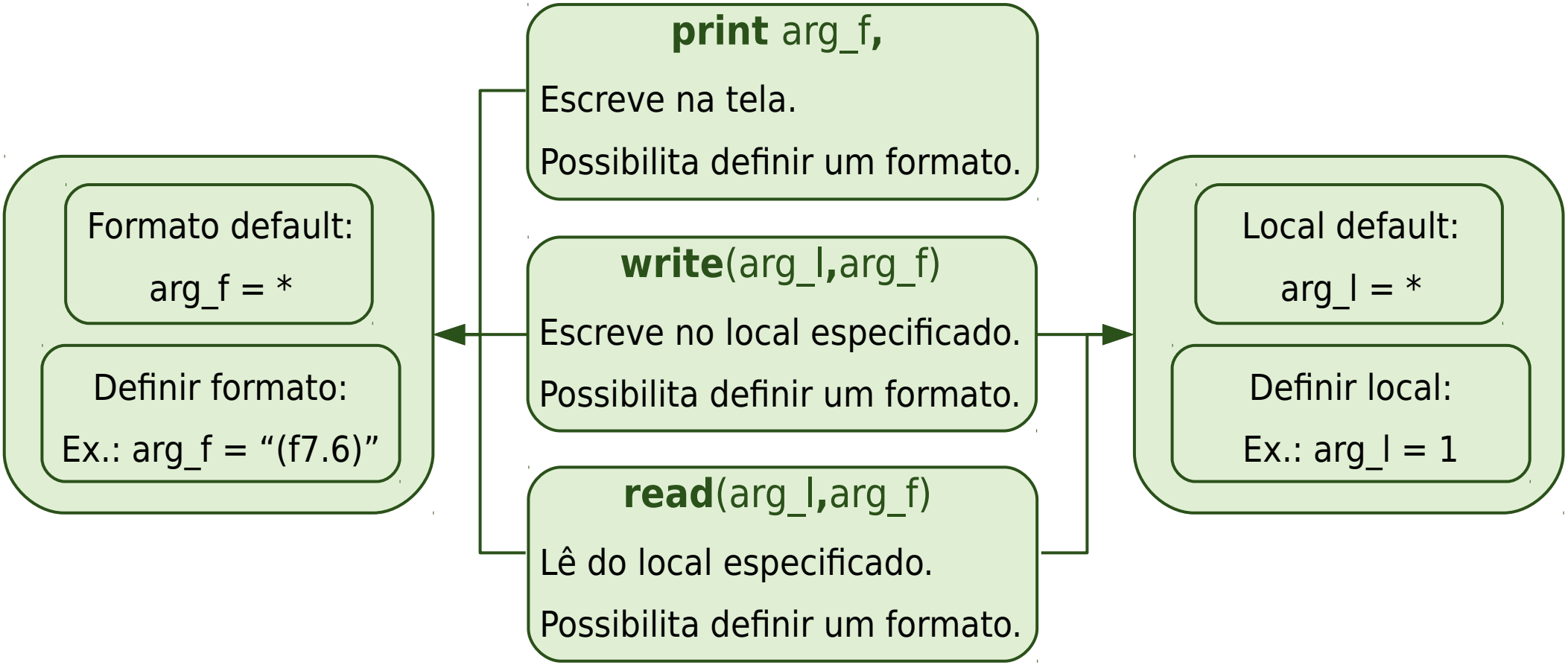
Operadores

Soma	Subtração	Multipicação	Divisão	Exponenciação
+	-	*	/	**

```
integer :: i_a, i_b
real(8) :: r_a, r_b
real(8) :: r_array(3)=1.0d0

i_a = 2
i_b = 5
r_a = 2.3d0
r_b = 5.2d0

print*, "r_a+r_b =",r_a+r_b
print*, "r_a-r_b =",r_a-r_b
print*, "r_a*i_a =",r_a*i_a
print*, "r_array/i_b =",r_array/i_b
print*, "i_a**i_b =",i_a**i_b
print*, "i_b/i_a =",i_b/i_a
print*, "r_a**(i_b/i_a) =",r_a**(i_b/3.d0)
```

```
r_a+r_b =   7.5000000000000000
r_a-r_b =  -2.9000000000000004
r_a*i_a =   4.5999999999999996
r_array/i_b =  0.20000000000000001       0.20000000000000001
     0.20000000000000001
i_a**i_b =           32
i_b/i_a =            2
r_a**(i_b/i_a) =   4.0075571716309755
```

Input/Output



Input/Output

Tabela de formatos	
Formato	Significado
in	Inteiro com “n” dígitos
fn.d	Real com “n” dígitos e “d” casas decimais
en.d	Real em notação científica
an	Character com “n” letras
n**x**	“n” espaços
/	Pula linha

O número “n” de dígitos deve incluir pontos (.) e sinal (-)

Input/Output

Escrevendo/lendo de um arquivo

1. Abra o arquivo:

   open(identificador integer**,FILE="**nome do arquivo**"** (string)**,ACTION=**read/write**)**

2. Escreva no arquivo... :

   write(identificador integer**,*)**

3. ou leia do arquivo:

   read(identificador integer**,*)**

4. Por fim, feche o arquivo

   close(identificador integer**)**

Você pode escrever um integer a sua escolha no identificador ou pode escrever "**NEWUNIT=**variável integer" para que um inteiro seja gerado automaticamente na variável integer fornecida.

Loops (Laços) – Comando “do”

do i = N_0 , N

bloco de código

enddo

Execute o “bloco de código” para “i” variando de N_0 até N.

```
integer :: i, soma
integer :: r(6)=[2,4,6,8,10,12]

soma = 0
do i = 1,6
    soma = soma + r(i)
enddo
```

$$soma = \sum_{i=1}^{6} r(i)$$

```
soma = 42
```

Loops (Laços) – Comando “do”

Nested loops

(laços aninhados)

do i = N_0 , N

  do j = n_0 , n

    bloco de código

  enddo

enddo

$$\text{prod} = \sum_{i=1}^{6} \sum_{j=1}^{4} r(i)\,b(j)$$

```
integer :: i, j prod
integer :: r(6)=[2,4,6,8,10,12]
integer :: b(4)=[1,3,5,7]

prod = 0
do i = 1,6
    do j = 1,4
        prod = prod + r(i)*b(j)
    enddo
enddo
```

```
prod = 672
```

Loops (Laços) – Comando “do”

Obs 1: Em arrays bidimensionais ou maiores, os elementos são acessados mais rapidamente nos índices mais a esquerda, ou seja, dê preferência a índices mais a esquerda em loops internos.

Isso está relacionado com o apresentado no slide 24

mais lento

```
!c(4,3)
soma = 0
do i = 1,4
    do j = 1,3
        soma = c(i,j)
    enddo
enddo
```

mais rápido

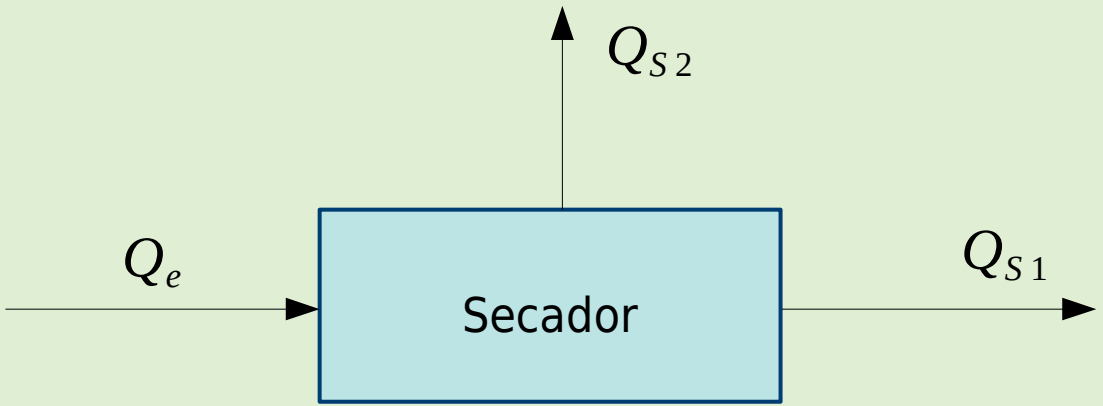
```
!c(4,3)
soma = 0
do j = 1,3
    do i = 1,4
        soma = c(i,j)
    enddo
enddo
```

Obs 2: Para ler/escrever um número N de elementos numa mesma linha, faz-se um loop acoplado ao comando read/write.

```
read(input,*) (x(i),i=1,N)
```

Exercício 1: Balanço de massa

Seja o processo de secagem abaixo, que separa água de partículas sólidas.



Sabendo que a corrente Q_{S2} possui apenas água, varie as frações de água e sólidos na entrada e faça um gráfico as relacionando com as frações na saída S1. Dados: Q_e=100kg/h e η=70%

Equações de balanço de massa:

$$\eta = \frac{Q_{S2}}{Q_e x^e_{H_2O}}$$

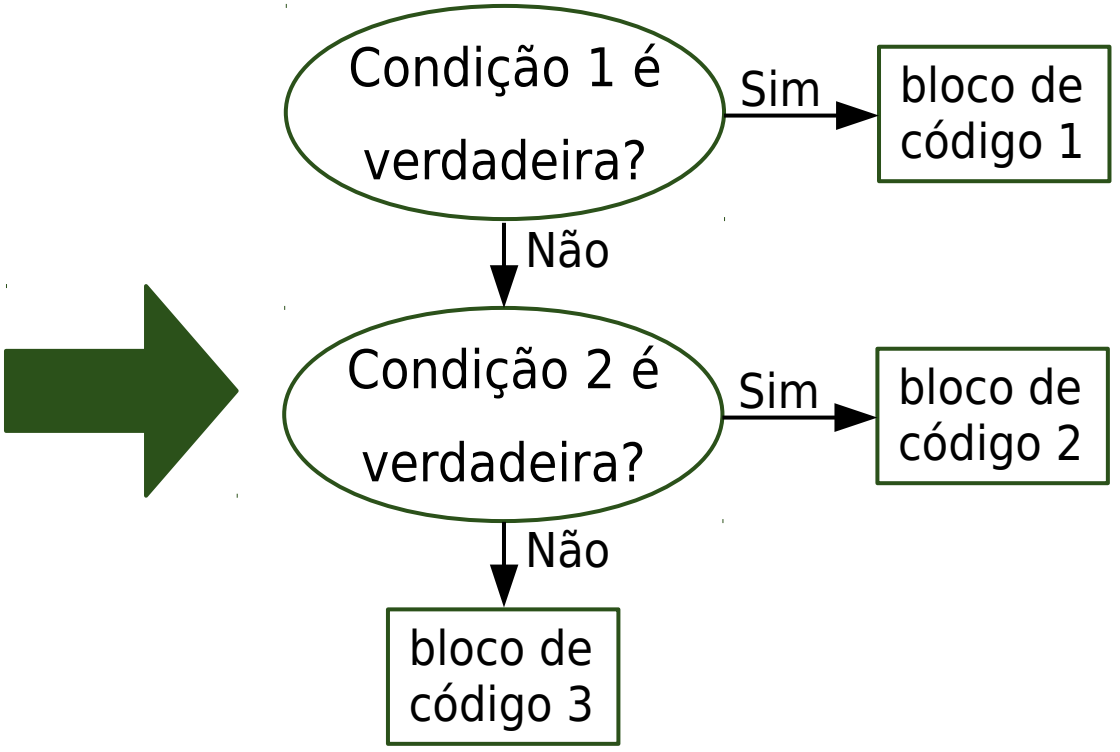
$$Q_{S1} = Q_e - Q_{S2}$$

$$Q_e x^e_{H_2O} = Q_{S2} + Q_{S1} x^{S1}_{H_2O}$$

$$Q_e x^e_{sólido} = Q_{S1} x^{S1}_{sólido}$$

Condicional – Comando “if”

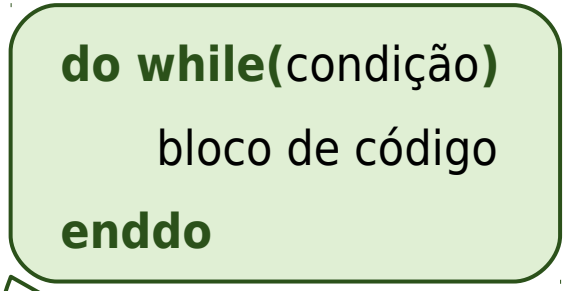
```
If(condição 1) then
        bloco de código 1
else if (condição 2) then
        bloco de código 2
else
        bloco de código 3
endif
```

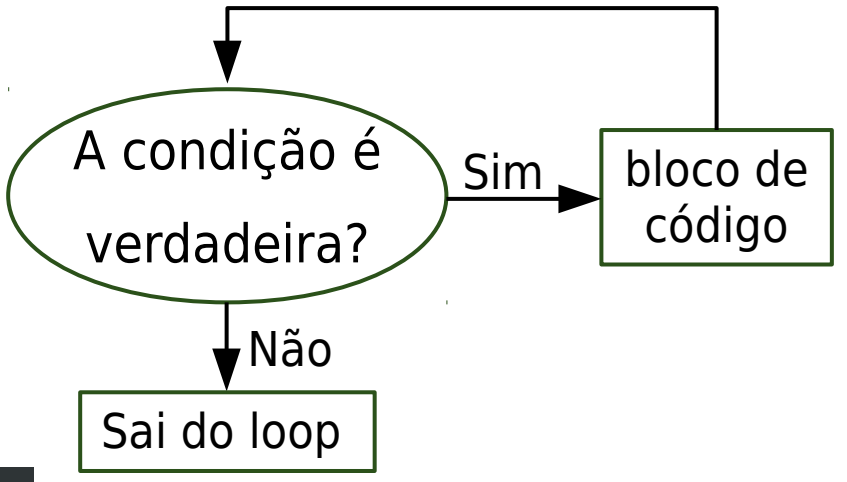


Condicional – Comando “if”

Tabela de formatos		
Símbolo	Significado	Símbolo alternativo
==	igual	**.eq.**
/=	diferente	**.ne.**
>	Maior que	**.gt.**
>=	Maior ou igual a	**.ge.**
<	Menor que	**.lt.**
<=	Menor ou igual a	**.le.**
.and.	e	Não tem
.or.	ou	Não tem

Loop condicional – Comando “do while”






```
f = 2.45d0 !f é um real(8)
do while(f > 1.d-6)
    print*, f
    f = f/4.d0
enddo
print*, f
```

```
 2.4500000000000002
0.61250000000000004
0.15312500000000001
 3.8281250000000003E-002
 9.5703125000000007E-003
 2.3925781250000002E-003
 5.9814453125000004E-004
 1.4953613281250001E-004
 3.7384033203125003E-005
 9.3460083007812507E-006
 2.3365020751953127E-006
 5.8412551879882817E-007
```

Exercício 2: Cálculo de derivada

Calcular a derivada da função abaixo em x=2,8334.

$$f(x) = \ln(1{,}23478\,x^3 + 2{,}123\,x^2 - 3{,}7436\,x + 12{,}86457)$$

Derivada numérica:

$$\frac{df(x)}{dx} = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

Derivada analítica:

$$\frac{df(x)}{dx} = \frac{3{,}70434\,x^2 + 4{,}246\,x - 3{,}7436}{1{,}23478\,x^3 + 2{,}123\,x^2 - 3{,}7436\,x + 12{,}86457}$$

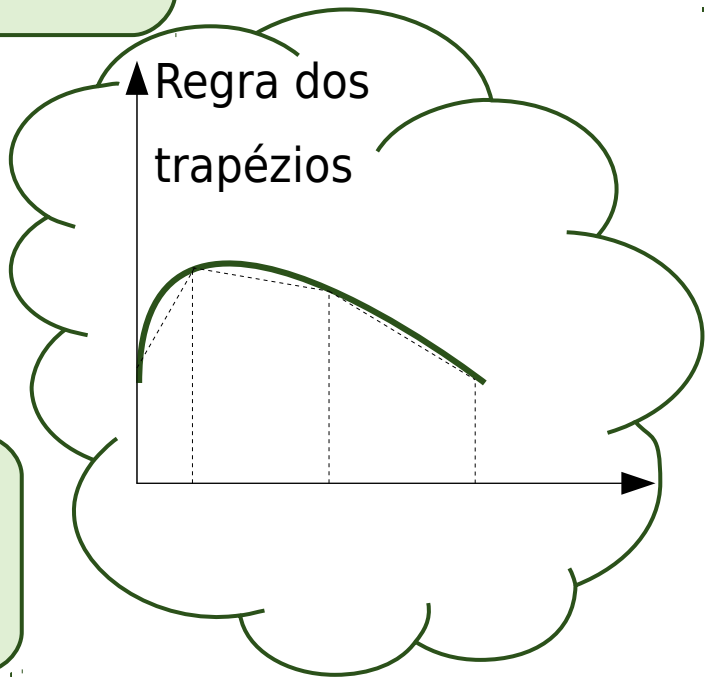
Exercício 3: Cálculo de integral

Calcular a integral da função entre 0 e 1.

$$f(x) = 4{,}2\,x^2 - 1{,}2\,x + 5{,}1$$

Integral numérica:

$$\int f(x) \approx \sum_{i=0}^{i=N} \left(\frac{f(x_i) + f(x_i + \Delta x)}{2} \Delta x \right)$$



Integral analítica:

$$\int f(x) = 1{,}4\,x^3 - 0{,}6\,x^2 + 5{,}1\,x$$

Exercício 4: Cálculo de T_{sat} dado P

Um cálculo comumente realizado em problemas de equilíbrio de fases consiste em obter a temperatura de saturação (T_{sat}) a uma pressão dada.

Para isso, normalmente é preciso aplicar o método das substituições sucessivas na equação de Wagner.

Equação de Wagner

$$\ln\left(\frac{P_{sat}}{P_c}\right)=\frac{T_c}{T}\left(a\,\tau+b\,\tau^{1,5}+c\,\tau^{2,5}+d\,\tau^{5}\right)$$

$$\tau=\left(1-\frac{T}{T_c}\right)$$

Problema:

Calcule a T_{sat} do butano a uma pressão de 2,8 bar.

Dados: Tc=425,25 K; a=-7,01763; b=1,6777; c=-1,9739; d=-2,172; Pc=37,92 bar.

Exercício 4: Cálculo de T_{sat} dado P

Para resolver esse problema da forma solicitada, deve-se colocar T em evidência, como no exemplo abaixo.

A partir de uma estimativa inicial para T, calcula-se um novo T. Este é então inserido do lado dirieto da equação e um novo T é calculado novamente. O processo deve ser repetido até que T varie pouco.

$$T = \frac{T_c\left(a\,\tau + b\,\tau^{1,5} + c\,\tau^{2,5} + d\,\tau^{5}\right)}{\ln\left(\frac{P_{sat}}{P_c}\right)}$$

$$\tau = \left(1 - \frac{T}{T_c}\right)$$

Não deu certo, né?

Problema:

Calcule a T_{sat} do butano a uma pressão de 2,8 bar.

Dados: Tc=425,25 K; a=-7,01763; b=1,6777; c=-1,9739; d=-2,172; Pc=37,92 bar.

Exercício 4: Cálculo de T_{sat} dado P

Não deu certo, né?

Apareceu “NaN”? É isso mesmo.

NaN (Not a Number) é um dos erros mais comuns em programação.

Ele occore quando são realizadas contas que o computador não consegue processar. Assim, devemos associá-lo a um dos seguintes motivos:

- **Log de zero ou número negativo;**
- **Raíz de ordem par de número negativo;**
- **Divisão por zero.**

Dependendo da equação com que estamos trabalhando e do método numérico empregado, é comum este não convergir, mas sim divergir. processo deve ser repetido até que T varie pouco.

Exercício 4: Cálculo de T_{sat} dado P

Tente utilizar essa forma alternativa, na qual foi multiplicado T em ambos os lados da equação anterior e, em seguida, tirou-se a raiz quadrada da expressão inteira.

$$T = \sqrt{\frac{T_c\left(a\,\tau + b\,\tau^{1,5} + c\,\tau^{2,5} + d\,\tau^{5}\right)T}{\ln\left(\frac{P_{sat}}{P_c}\right)}}$$

$$\tau = \left(1 - \frac{T}{T_c}\right)$$

Problema:

Calcule a T_{sat} do butano a uma pressão de 2,8 bar.

Dados: Tc=425,25 K; a=-7,01763; b=1,6777; c=-1,9739; d=-2,172; Pc=37,92 bar.

Funções e Subrotinas

Atenção:

Existem diferentes formas de adicionar uma subrotina ou função a um programa.

As mais importantes são:

- Escrever “***contains***” dentro do “***program***” e adicionar as funções/subrotinas embaixo.
- Utilizar **módulos**.

Funções e Subrotinas

Funções

function nome_da_func**(**argumento1**, ...)**

Declaração de variáveis

Cálculos da função

end function

```
function area_circulo(r)
    implicit none

    !Argumento
    real(8) :: r
    !Local
    real(8), parameter :: pi = 3.1415d0
    !Retorno
    real(8) :: area_circulo

    area_circulo = pi*r**2
end function
```

Como chamar funções

```
!Fluxo volumétrico
Q = area_circulo(r)*vel
```

ou

```
A = area_circulo(r)
!Fluxo volumétrico
Q = A*vel
```

A função retorna uma variável com o mesmo nome da função.

Funções e Subrotinas

Funções - Alternativa

function nome_da_func**(**argumento1**, ...) result(**var_retorno**)**

    Declaração de variáveis

    Cálculos da função

end function

Nesse caso, definimos qual é a variável de retorno.

```
function area_circulo(r) result(area)
    implicit none

    !Argumento
    real(8) :: r
    !Local
    real(8), parameter :: pi = 3.1415d0
    !Retorno
    real(8) :: area

    area = pi*r**2
end function
```

Funções e Subrotinas

Subrotinas

subroutine nome_da_sub**(**argumento1**, ...)**

Declaração de variáveis

Cálculos da função

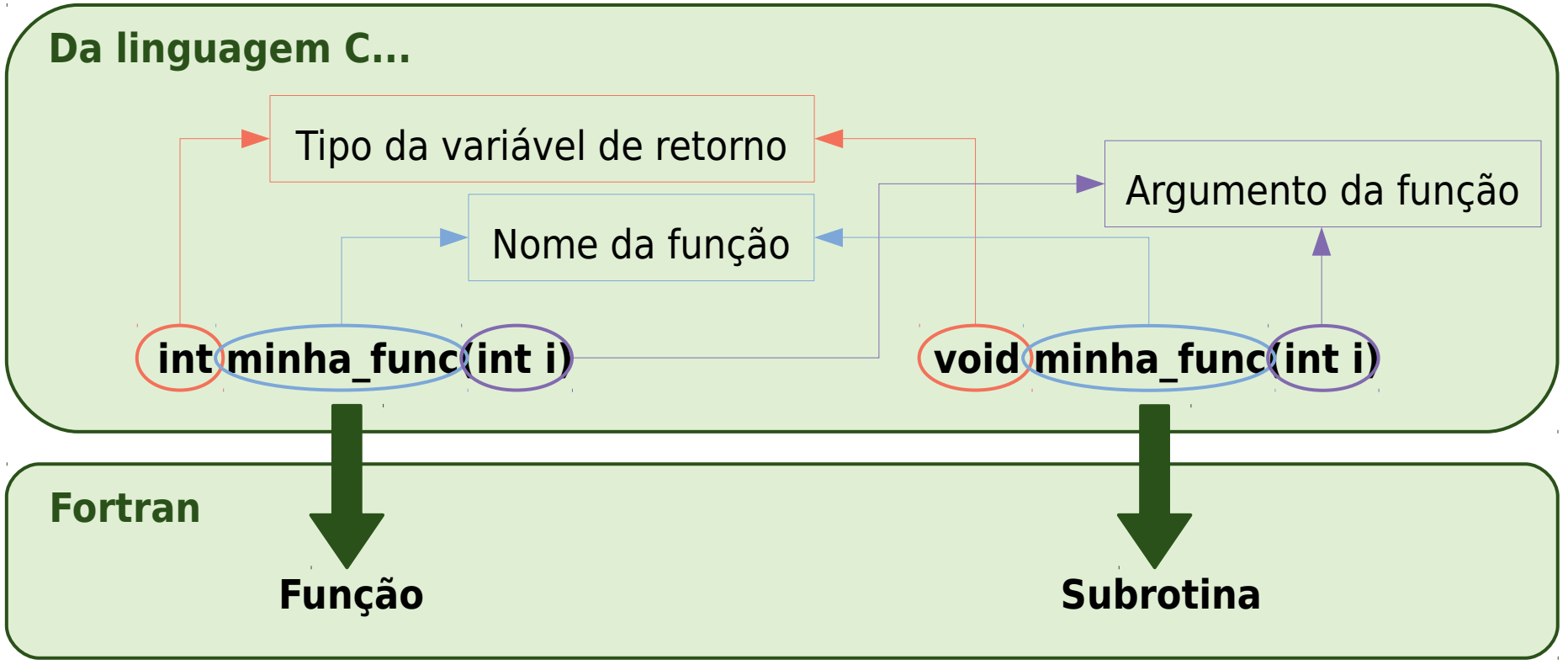
end subroutine

```
subroutine area_circulo(r, area)

    implicit none

    !Arguments
    real(8) :: r, area
    !Local
    real(8), parameter :: pi = 3.1415d0

    area = pi*r**2
end subroutine
```

Como chamar subrotinas

```
call area_circulo(r,area)
!Fluxo volumétrico
Q = area*vel
```

Funções e Subrotinas



Obs: Em C, uma função do tipo void é uma função que não retorna nenhuma variável.

Funções e Subrotinas

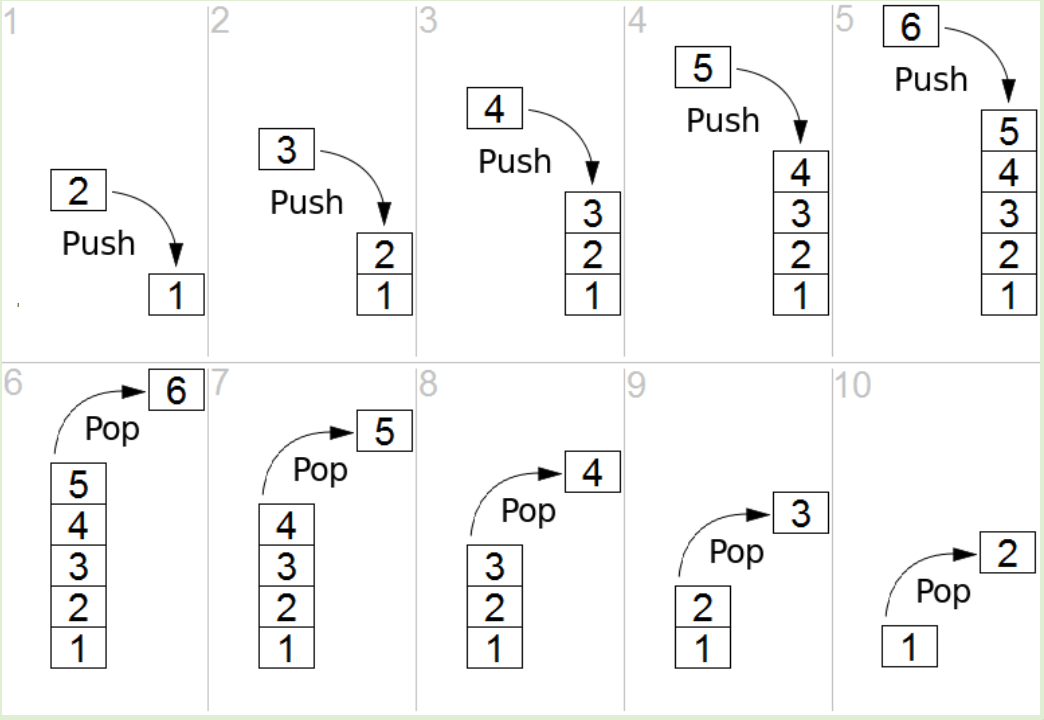
Observação:

Quando uma função, subrotina ou programa é inicializado, é criado um **stack**.

O stack é uma lista de variáveis e chamadas de função.

LIFO – Last In First Out

Como funciona o stack



Funções e Subrotinas

Stacks na prática:

- Acesse o link:

http://www.pythontutor.com/c.html#mode=edit

- Abra o arquivo “seq_C_stack.c”, copie o código presente nele e substitua o código presente no link por esse. Esse código possui uma pequena rotina em C composta pelo programa principal e outras duas funções.
- Clique em “Visualize Execution” e espere alguns segundos.
- Clique seguidamente em “forward” e observe a dinâmica de formação dos stacks.

Funções e Subrotinas

Observação:

Passagem por valor ✕ **Passagem por referência**

Fortran faz assim

A função tem acesso somente à “aparência” da variável. Com isso, ela cria uma cópia local da variável. Assim, qualquer modificação realizada na variável não irá afetar a variável original fora da função.

A função tem acesso ao endereço da variável na memória. Assim, as modificações nela feitas serão permanentes. Apesar de menos segura, essapassagem é mais rápida, pois dispensa a criação de uma cópia da variável.

Funções Recursivas

```
recursive function nome_da_func(argumento1, ...)
    Declaração de variáveis


    Cálculos da função
end function
```

A função chama ela própria.

```
recursive function soma_pa(a1,n_termos,razao,i) result(soma)
    !Argumento
    real(8) :: a1, razao
    integer :: n_termos, i
    !Retorno
    real(8) :: soma

    if(i .le. n_termos) then
        soma = (a1+razao*(i-1)) + soma_pa(a1,n_termos,razao,i+1)
    endif
end function
```

Exercício 5: Cálculo do fatorial

Utilize funções recursivas para calcular o fatorial de um número inteiro dado pelo usuário.

Arrays dinâmicos

Alocação dinâmica de memória

- **Declaração de variável:**

tipo, allocatable :: variável1(:), variável2(:,:)

- **Alocação da variável:**

allocate(variável1(N),variável2(N,N))

- **Desalocação da variável:**

deallocate(variável)

Heap (local da memória)

- **alocação dinâmica de variáveis;**
- **Tempo de execução (*runtime*);**
- **Sem padrão de alocação;**
- **Mais lento que o stack;**
- **Mais espaço que o stack;**

Exercício 6: Algoritmo para ordenar (*sort*)

Seja um array unidimensional de números reais, o qual tanto o tamanho quanto os seus elementos são dados pelo usuário por meio de um arquivo de entrada.

Desenvolver um programa para criar o array e ordená-lo em ordem crescente.

Módulos

- Consistem em “pacotes”, que podem armazer variáveis e funções/subrotinas;
- Possibilitam dividir o programa em diferentes arquivos;
  - Podem ser criados em diferentes arquivos ou no mesmo arquivo do **program**.

module nome_do_módulo

Bloco de declaração de variáveis

contains

Bloco de declaração de funções e subrotinas

end module

```
module meu_modulo
    implicit none
    integer :: iii = 1
contains
    subroutine sub1(a)
        real(8) :: a
        print*, (a + iii)
    end subroutine
end module
```

Escopo de variáveis

Variáveis locais: As variáveis criadas dentro de uma função ou subrotina são de escopo local, ou seja, elas existem apenas dentro da respectiva função ou subrotina. Elas não são enxergadas em nenhuma outra parte do programa.

Obs: As funções e subrotinas inseridas embaixo do **contains** de um **program** enxergam todas as variáveis declaradas nele.

Escopo de variáveis

Variáveis globais: As variáveis declaradas na parte superior do **contains** de um módulo são globais em todos os locais onde o módulo for usado.

Obs:

```
module meu_modulo
    implicit none

    integer :: um_int = 1
contains
```

→ Essa variável pode ser acessada a partir de qualquer arquivo que esteja usando o módulo onde ela foi declarada.

```
module meu_modulo
    implicit none

    integer, private :: um_int = 2
contains
```

→ Essa variável só pode ser acessada pelas funções e subrotinas localizadas abaixo do contains do mesmo módulo.

Funções intrínsecas (*built in functions*)

Sum() = retorna o somatório de um array.

Max() = retorna o maior valor entre os números avaliados.

Min() = retorna o menor valor entre os números avaliados.

Maxval() = retorna o maior valor de uma array.

Minval() = retorna o menor valor de uma array.

Abs() = retorna o módulo do número avaliado.

Exp() = retorna a exponencial do número avaliado.

Log() = retorna o log neperiano do número avaliado.

Log10() = retorna o log do base 10 do número avaliado.

Nint() = aproxima o número avaliado para um inteiro.

Matmul() = Realiza multiplicação matricial (produto interno).

Exercício 7: Método de Euler explícito

A EDO (Equação Diferencial Ordinária) ao lado representa o consumo de um componente A ao longo do tempo de acordo com uma cinética de reação de segunda ordem. Resolva a EDO utilizando o método de Euler explícito para realizar a integração temporal.

Dados:

Para t=0, C_A = 1 mol/L

t = 0,54 L/(mol.s) (constante cinética)

h = 1 segundo (passo de tempo)

t_{final} = 2 minutos

EDO:

$$\frac{dC_A}{dt} = -k.C_A^{\,2}$$

Método de Euler explícito:

$$y_{i+1} = y_i + h.f(t_i, y_i)$$

Aplicando o método na EDO:

$$C_{A_{i+1}} = C_{A_i} - h.k.C_A^{\,2}$$

“Debugando” um programa

Breakpoint: Sinalizam onde o programa deve pausar. Para criar um, basta clicar do lado direito do número da linha onde deseja criá-lo.

Rodar o Debugger: Clique na seta vermelha localizada na barra de ferramentas superior.

Rasterar variáveis: clique em “**Debug**”, localizado na barra superior do CODE::BLOCKS. Em seguida, clique em “**debugging windows**” e, depois, em “**watches**”. Com isso, é possível monitorar o valor das variáveis ao longo do programa

Observação: Comandos usados comumente para ajudar a debugar manualmente o programa.

Stop: Pára o programa.

Pause: Pausa o programa até o usuário intervir.

Melhores amigos do programador