

## Introdução à Programação de Computadores

### Variáveis

# Meu segundo programa

- Uma conta poupança foi aberta com um depósito de R\$500,00, com rendimentos 1% de juros ao mês. No segundo mês, R\$200,00 reais foram depositados nessa conta poupança. No terceiro mês, R\$50,00 reais foram retirados da conta. Quanto haverá nessa conta no quarto mês?

# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

endereço	conteúdo
Rua do Ouro, 12	Edifício Luz
Rua do Ouro, 13	Casa da Maria
Rua do Ouro, 14	Padaria do Zé
Rua do Ouro, 15	Farmácia Legal
Rua do Ouro, 16	Casa do João
Rua do Ouro, 17	Edifício do Sol

# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

Cada gaveta tem uma etiqueta e um espaço bem delimitado. No entanto, você pode guardar diversas coisas dentro delas.



# Variáveis

- Os dados que um programa utiliza precisam ser armazenados na **memória** do computador.
- Cada posição de memória do computador possui um **endereço**.

endereço	conteúdo
6612	891
6613	'a'
6614	8
6615	16.1
6616	0.4543
6617	298347

# Variáveis

- A partir dos endereços, é possível para o computador saber qual é o valor armazenado em cada uma das posições de memória.
- Como a **memória pode ter bilhões de posições**, é difícil controlar em qual endereço está armazenado um determinado valor!
- Para facilitar o controle sobre onde armazenar informação, os programas utilizam **variáveis**.
- Uma variável corresponde a um **nome simbólico (ou etiqueta)** de uma posição de memória.
- Seu **conteúdo pode variar** durante a execução do programa.

# Variáveis

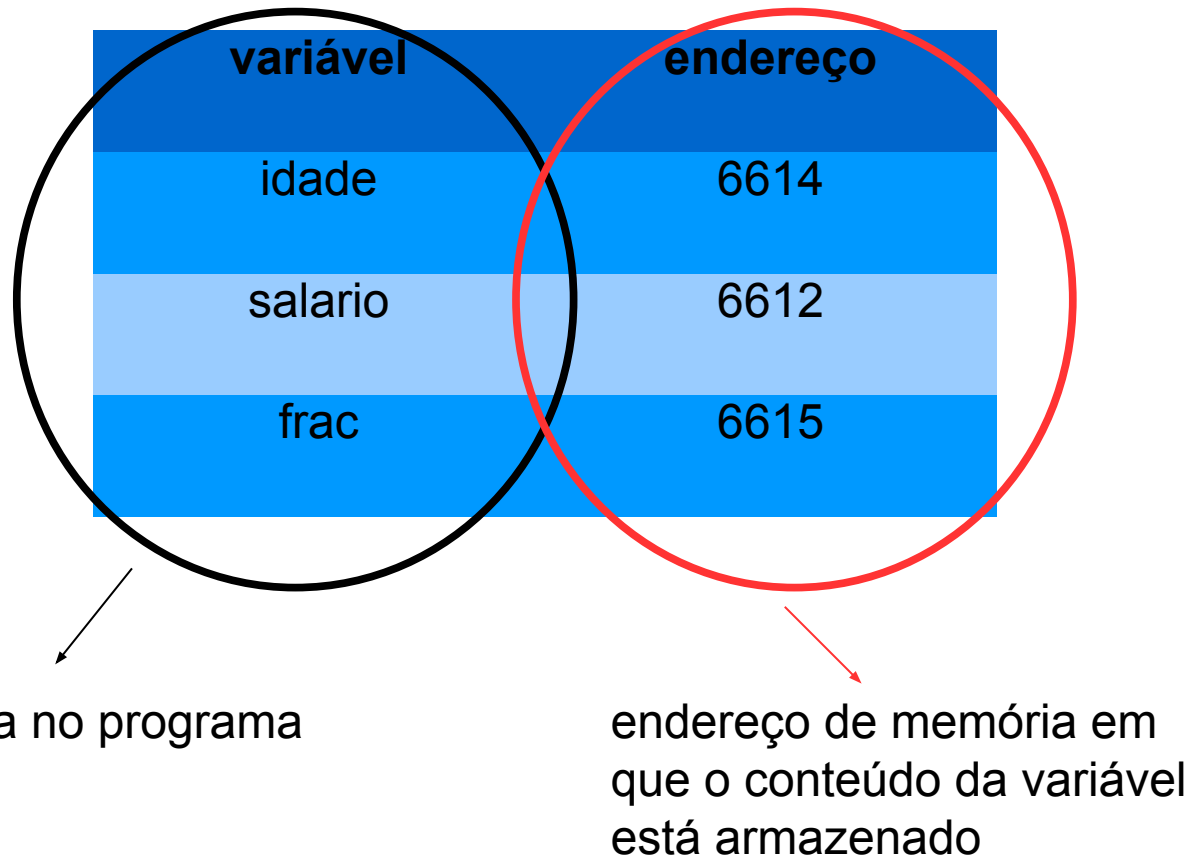
- Dicionário de variáveis do compilador

variável	endereço
idade	6614
salario	6612
frac	6615



# Variáveis

- Dicionário de variáveis



# Variáveis

## Dicionário de variáveis

<b>variável</b>	<b>endereço</b>
idade	6614
salario	6612
frac	6615



## Memória do computador

<b>endereço</b>	<b>conteúdo</b>
6611	9439.23496
6612	891
6613	'P'
6614	8
6615	0.4543
6616	2365

# Variáveis

- Memória + dicionário de variáveis
- (vamos usar esta representação ao longo do curso!)

endereço	variável	conteúdo
6612	salario	891
6613	c	'a'
6614	idade	8
6615	velocidade	16.1
6616	frac	0.4543
6617	km	298347

# Variáveis

- Exemplo de variável:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

A variável **y** irá armazenar o valor de **sin(1.5)**.

# Variáveis

- Cada variável pode possuir uma quantidade diferente de bytes, uma vez que os tipos de dados são representados de forma diferente.
- Portanto, a cada variável está associado um tipo específico de dados.
- Logo:
  - O tipo da variável define quantos bytes de memória serão necessários para representar os dados que a variável armazena.

# Variáveis

- A Linguagem C dispõe de **quatro tipos básicos de dados**. Assim, as variáveis poderão assumir os seguintes tipos:

tipo	tamanho (bytes)	valor
char	1	Um caractere ou um inteiro de 0 a 127
int	4	um número inteiro
float	4	um número de ponto flutuante (SP)
double	8	um número de ponto flutuante (DP)

# Variáveis

- Dentro do programa, as variáveis são identificadas por seus **nomes**.
- Portanto, um programa deve **declarar** todas as variáveis que irá utilizar.
- Atenção!
  - A declaração de variáveis deve ser feita antes que a **variável seja usada**, para garantir que a quantidade correta de memória já tenha sido reservada para armazenar seu valor.

# Variáveis

- Para assinalar valores à variáveis deve-se usar o **operador de atribuição =**

nome\_da\_variavel = (expressão);

 a expressão é calculada e depois atribuída à variável

Exemplos:

float bonus, salario, conta;

bonus = 0.01;

salario = 985.83;

conta = salario \* juros;



# Variáveis

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

```
void main(void) {  
    int idade;  
    float salario;  
    char sexo;  
    double divida;  
  
    idade = 25;  
    salario = 100.5;  
    sexo = 'M';  
    divida = 29999.99;  
  
    printf("Eu tenho %d anos,", idade);  
    printf(" recebo %f reais por mes,", salario);  
    printf(" sou do sexo %c", sexo);  
    printf(" e tenho uma divida de %f", divida);  
    printf("\n");  
    system("PAUSE");  
}
```

Endereço	Variável	Conteúdo
4812		
4813		
4814		
4815		
4816		
4817		
4818		
4819		

# Variáveis

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

```
void main(void) {
```

```
    int idade;
```

```
    float salario;
```

```
    char sexo;
```

```
    double divida;
```

```
    idade = 25;
```

```
    salario = 100.5;
```

```
    sexo = 'M';
```

```
    divida = 29999.99;
```

```
    printf("Eu tenho %d anos,", idade);
```

```
    printf(" recebo %f reais por mes,", salario);
```

```
    printf(" sou do sexo %c", sexo);
```

```
    printf(" e tenho uma divida de %f", divida);
```

```
    printf("\n");
```

```
    system("PAUSE");
```

```
}
```

Endereço	Variável	Conteúdo
4812	idade	
4813	salario	
4814	sexo	
4815	divida	
4816		
4817		
4818		
4819		

# Variáveis

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

```
void main(void) {
```

```
    int idade;
```

```
    float salario;
```

```
    char sexo;
```

```
    double divida;
```

```
    idade = 25;
```

```
    salario = 100.5;
```

```
    sexo = 'M';
```

```
    divida = 29999.99;
```

```
    printf("Eu tenho %d anos,", idade);
```

```
    printf(" recebo %f reais por mes,", salario);
```

```
    printf(" sou do sexo %c", sexo);
```

```
    printf(" e tenho uma divida de %f", divida);
```

```
    printf("\n");
```

```
    system("PAUSE");
```

```
}
```

Endereço	Variável	Conteúdo
4812	idade	25
4813	salario	100.5
4814	sexo	'M'
4815	divida	29999.99
4816		
4817		
4818		
4819		

# Escrevendo um programa em C

- A primeira linha do corpo da função principal do programa `p1.c` é:

```
float y;
```

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Escrevendo um programa em C

- Declarando duas ou mais variáveis do mesmo tipo

```
float y, aux, salario;
```

# Escrevendo um programa em C

- Esta linha declara uma variável *y* para armazenar um número de ponto flutuante (SP).
- A declaração de uma variável não armazena valor algum na posição de memória que a variável representa.
- Ou seja, no caso anterior, vai existir uma posição de memória chamada *y*, mas ainda não vai existir valor armazenado nesta posição.

# Escrevendo um programa em C


- Um valor pode ser **atribuído** a uma posição de memória representada por uma variável pelo **operador de atribuição =**.
- O operador de atribuição requer à esquerda um nome de variável e à direita, um valor.
- A linha seguinte de **p1.c** atribui um valor a **y**:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

# Escrevendo um programa em C

- No lado direito do operador de atribuição existe uma referência à função **seno** com um parâmetro **1.5** (uma constante de ponto flutuante representando um valor em **radianos**.)

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```





# Escrevendo um programa em C

- Em uma linguagem de programação chamamos o valor entre parênteses da função, neste exemplo, o valor 1.5, de **parâmetro da função**.
- Da mesma forma, diz-se que **sin(1.5)** é o valor da **função sin** para o parâmetro **1.5**.
- O operador de atribuição na linha **y = sin(1.5)** obtém o valor da função (0.997495) e o armazena na posição de memória identificada pelo nome **y**.
- Esta operação recebe o nome de: **atribuição de valor a uma variável**.

# Escrevendo um programa em C

- Atenção: O valor armazenado em uma variável por uma operação de atribuição depende do tipo da variável.
- Se o tipo da variável for `int`, será armazenado um valor inteiro (caso o valor possua parte fracionária, ela será desprezada).
- Se o tipo da variável for `float` ou `double`, será armazenado um valor de ponto flutuante (caso o valor não possua parte fracionária, ela será nula).

# Escrevendo um programa em C

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

Endereço	Variável	Conteúdo
8512		
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

Endereço	Variável	Conteúdo
8512	<b>y</b>	
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

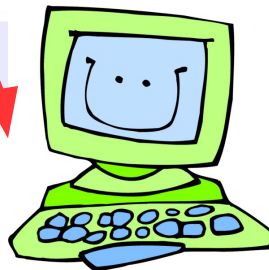
```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

Endereço	Variável	Conteúdo
8512	<b>y</b>	
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

**sin(1.5)**



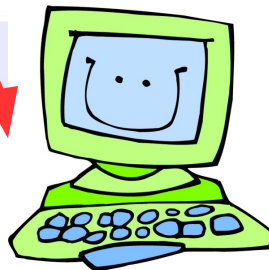
Endereço	Variável	Conteúdo
8512	<b>y</b>	
8513		
8514		
8515		
8516		
8517		
8518		
8519		

(processador)

# Escrevendo um programa em C

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char* argv[]) {
5     float y;
6     y = sin(1.5);
7     printf("seno de 1.5 eh: %f", y);
8     printf("\n");
9     system("PAUSE");
10    return 0;
11 }
```

**sin(1.5)**



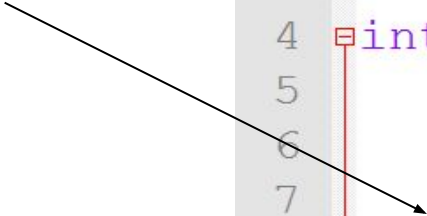
(processador)

Endereço	Variável	Conteúdo
8512	<b>y</b>	<b>0.997495</b>
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

- As próximas linhas do programa `p1.c` são:

```
printf("y = %f", y);  
printf("\n");
```



```
1  #include <stdio.h>  
2  #include <math.h>  
3  
4  int main(int argc, char* argv[]) {  
5      float y;  
6      y = sin(1.5);  
7      printf("seno de 1.5 eh: %f", y);  
8      printf("\n");  
9      system("PAUSE");  
10     return 0;  
11 }
```

- A função `printf` faz parte da biblioteca `stdio`.



# Escrevendo um programa em C

- A função `printf` é usada para exibir resultados produzidos pelo programa e **pode ter um ou mais parâmetros**.
- O primeiro parâmetro da função `printf` é sempre uma **string**, correspondente à sequência de caracteres que será exibida pelo programa.

```
printf("y = %f", y);  
printf("\n");
```

# Escrevendo um programa em C

- Essa sequência de caracteres pode conter algumas **tags** que representam valores, conhecidas como **especificadores de formato**.

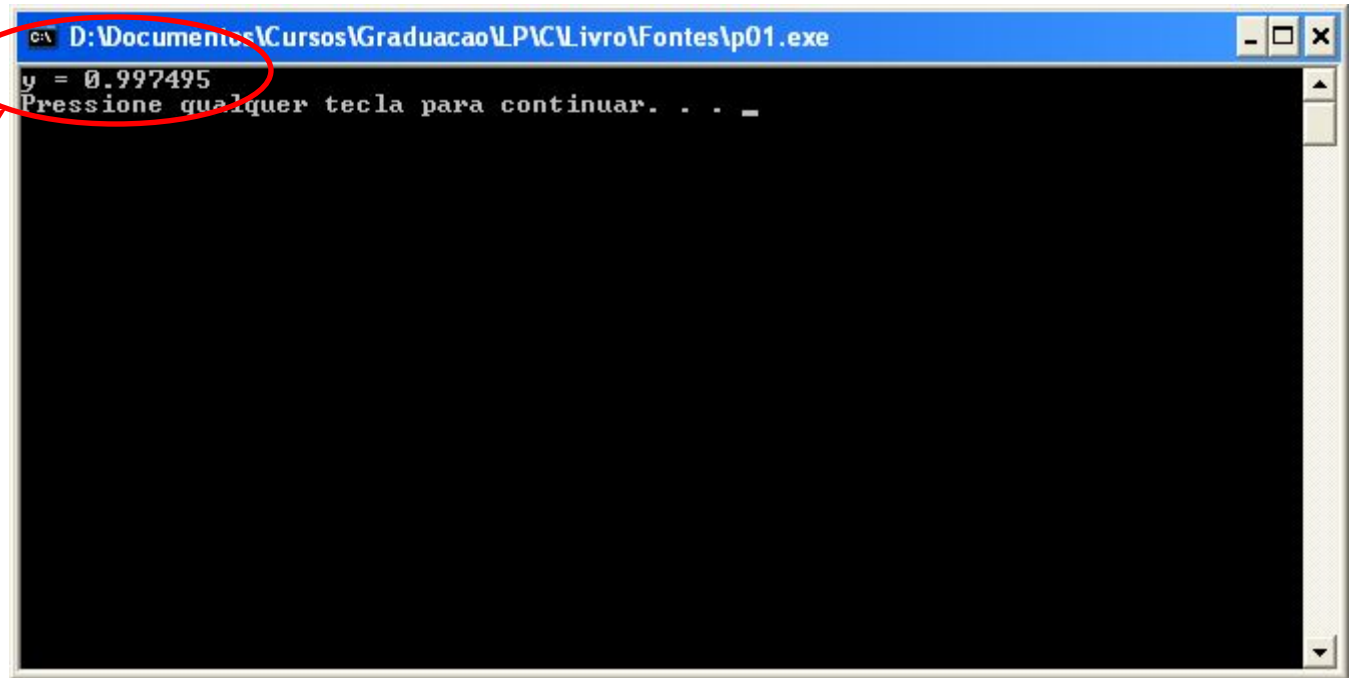
```
printf("y = %f", y);  
printf("\n");
```

Especificador  
de formato

- Um especificador de formato começa sempre com o símbolo **%**. Em seguida, pode apresentar uma **letra** que indica o tipo do valor a ser exibido.
- Assim, **printf("y = %f", y)** irá exibir a letra **y**, um espaço em branco, o símbolo **=**, um espaço em branco, e um valor de ponto flutuante.

# Escrevendo um programa em C

Veja:



```
C:\> D:\Documentos\Cursos\Graduacao\LP\CLivro\Fontes\lp01.exe
y = 0.997495
Pressione qualquer tecla para continuar. . . _
```

Valor  
armazenado  
em *y*.

# Escrevendo um programa em C

- Na função `printf`, para cada `tag` existente no primeiro parâmetro, deverá haver um novo parâmetro que especifica o valor a ser exibido.

```
printf("a = %d, b = %c e c = %f", a, 'm', (a+b));
```

# Formatação de valores numéricos

- Além de especificar o número de casas decimais, um **tag** pode especificar o número total de caracteres (incluindo o sinal e o ponto decimal).
- Assim, o tag **%8.3f** significa: “exibir um valor de ponto flutuante com oito caracteres no total e com três casas decimais”.
- Se for necessário, será acrescentado o caractere ‘ ‘ (espaço) à esquerda do valor para completar o tamanho total.

# Formatação de valores numéricos

- Exemplo:

Valor	Tag	Valor exibido
pi = 3.14159	%5.3f	3.142
	%8.3f	3.142
raio = 2.0031	%5.3f	2.003
	%.6f	2.003100
	%5.3f	12.586
	%6.3f	12.586
	%7.3f	12.586
	%e	1.258584e+001
	%E	1.258584E+001
	%12.3e	1.259e+001

# Formatação de valores numéricos

- A formatação de valores pode ser feita também para números inteiros.
- Exemplo:

Valor	Tag	Valor exibido
3	%d	3
	%5d	3
	%01d	3
	%05d	00003

# Escrevendo um programa em C

- A linguagem C utiliza o símbolo `\` (barra invertida) para especificar alguns caracteres especiais:

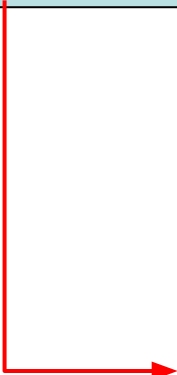
Caractere	Significado
<code>\a</code>	Caractere (invisível) de aviso sonoro.
<code>\n</code>	Caractere (invisível) de nova linha.
<code>\t</code>	Caractere (invisível) de tabulação horizontal.
<code>\'</code>	Caractere de apóstrofo



# Escrevendo um programa em C

- Observe a próxima linha do programa `p1.c`:

```
printf("\n");
```




```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

- Ela exibe “o caractere (invisível) de nova linha”. Qual o efeito disso? Provoca uma mudança de linha! Próxima mensagem será na próxima linha.

# Escrevendo um programa em C

- Observe agora a próxima linha do programa:

```
system("PAUSE");
```



```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```

- Ela exibe a mensagem “Pressione qualquer tecla para continuar...” e interrompe a execução do programa.

# Escrevendo um programa em C

- A execução será retomada quando o usuário pressionar alguma tecla.
- A última linha do programa `p1.c` é:

`return 0;`

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```


# Escrevendo um programa em C

- É usada apenas para satisfazer a sintaxe da linguagem C.
- O comando `return` indica o valor que uma função produz.
- Cada função, assim como na matemática, deve produzir um único valor.
- Este valor deve ter o mesmo tipo que o declarado para a função.

# Escrevendo um programa em C

- No caso do programa `p1.c`, a função principal foi declarada como sendo do tipo `int`. Ou seja, ela deve produzir um valor inteiro.

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(int argc, char* argv[]) {
5      float y;
6      y = sin(1.5);
7      printf("seno de 1.5 eh: %f", y);
8      printf("\n");
9      system("PAUSE");
10     return 0;
11 }
```



- A linha `return 0;` indica que a função principal irá produzir o valor inteiro 0.

# Escrevendo um programa em C

- Mas e daí?! O valor produzido pela função principal não é usado em lugar algum!
- Logo, não faz diferença se a última linha do programa for:

```
return 0;
```

```
return 1;
```

ou

```
return 1234;
```

# Escrevendo um programa em C

- Neste caso, o fato de a função produzir um valor não é relevante.
- Neste cenário, é possível declarar a função na forma de um **procedimento**.
- Um **procedimento** é uma função do tipo **void**, ou seja, uma função que produz o valor **void** (**vazio**, **inútil**, **à-toa**). Neste caso, ela não precisa do comando **return**.

# Escrevendo um programa em C

- Note que os parâmetros da função **main** também não foram usados neste caso.
- Portanto, podemos também indicar com **void** que a lista de parâmetros da função principal é vazia.
- Assim, podemos ter outras formas para **p1.c**:

```
void main(void)
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return;
}
```

```
void main(void)
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
}
```



# Avaliação de expressões aritméticas

- Os operadores aritméticos disponíveis na linguagem C são:

Operador	Operação
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão

# Conversão implícita de tipo

- Na avaliação de expressões aritméticas, estas operações são realizadas sempre entre operandos de mesmo tipo.
- Ou seja, o resultado da operação terá o mesmo tipo que os operandos.
- Caso haja valores inteiros e em ponto flutuante em uma expressão, haverá uma conversão implícita de tipo de `int` para `float`, sempre que necessário.

# Prioridade de execução das operações

- Porque as operações aritméticas devem ser feitas entre operandos do mesmo tipo?
  - As representações dos números inteiros e dos números de ponto flutuante são diferentes.
- Ou seja, embora 1 e 1.0 são valores iguais, eles têm representações diferentes no computador.
- **Prioridade de execução das operações:**
  - 1) expressões entre parênteses
  - 2) multiplicação, divisão e resto da divisão (da esquerda para a direita)
  - 3) operações de soma e subtração (da esquerda para a direita).

# Prioridade de execução das operações

- Exemplo:  $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja:  $a = 1.5$ ,  $b = 4$ ,  $c = 2$ ,  $d = 3$ ,  $e = 1.2$ ,  $f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
-------	----------	-----------	-------------------

# Prioridade de execução das operações

- Exemplo:  $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja:  $a = 1.5$ ,  $b = 4$ ,  $c = 2$ ,  $d = 3$ ,  $e = 1.2$ ,  $f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
1 <sup>a</sup>	$(c + d)$	$(2 + 3) = 5$	Não

# Prioridade de execução das operações

- Exemplo:  $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja:  $a = 1.5$ ,  $b = 4$ ,  $c = 2$ ,  $d = 3$ ,  $e = 1.2$ ,  $f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
1 <sup>a</sup>	$(c + d)$	$(2 + 3) = 5$	Não
2 <sup>a</sup>	$(e + f)$	$(1.2 + 4.3) = 5.5$	Não

# Prioridade de execução das operações

- Exemplo:  $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja:  $a = 1.5$ ,  $b = 4$ ,  $c = 2$ ,  $d = 3$ ,  $e = 1.2$ ,  $f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
1 <sup>a</sup>	$(c + d)$	$(2 + 3) = 5$	Não
2 <sup>a</sup>	$(e + f)$	$(1.2 + 4.3) = 5.5$	Não
3 <sup>a</sup>	$(a * 1^a)$	$(1.5 * 5) = 7.5$	Sim (5 para 5.0)

# Prioridade de execução das operações

- Exemplo:  $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja:  $a = 1.5, b = 4, c = 2, d = 3, e = 1.2, f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
1 <sup>a</sup>	$(c + d)$	$(2 + 3) = 5$	Não
2 <sup>a</sup>	$(e + f)$	$(1.2 + 4.3) = 5.5$	Não
3 <sup>a</sup>	$(a * 1^a)$	$(1.5 * 5) = 7.5$	Sim (5 para 5.0)
4 <sup>a</sup>	$(b * 2^a)$	$(4 * 5.5) = 22.0$	Sim (4 para 4.0)



# Prioridade de execução das operações

- Exemplo:  $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja:  $a = 1.5$ ,  $b = 4$ ,  $c = 2$ ,  $d = 3$ ,  $e = 1.2$ ,  $f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
1 <sup>a</sup>	$(c + d)$	$(2 + 3) = 5$	Não
2 <sup>a</sup>	$(e + f)$	$(1.2 + 4.3) = 5.5$	Não
3 <sup>a</sup>	$(a * 1^a)$	$(1.5 * 5) = 7.5$	Sim (5 para 5.0)
4 <sup>a</sup>	$(b * 2^a)$	$(4 * 5.5) = 22.0$	Sim (4 para 4.0)
5 <sup>a</sup>	$3^a / 4^a$	$7.5 / 22.0 = 0.341$	Não

# Prioridade de execução das operações

- Exemplo:  $v1 = (a * (c + d)) / (b * (e + f)) ;$

Seja:  $a = 1.5$ ,  $b = 4$ ,  $c = 2$ ,  $d = 3$ ,  $e = 1.2$ ,  $f = 4.3$

Ordem	Operação	Resultado	Conversão de tipo
1 <sup>a</sup>	$(c + d)$	$(2 + 3) = 5$	Não
2 <sup>a</sup>	$(e + f)$	$(1.2 + 4.3) = 5.5$	Não
3 <sup>a</sup>	$(a * 1^a)$	$(1.5 * 5) = 7.5$	Sim (5 para 5.0)
4 <sup>a</sup>	$(b * 2^a)$	$(4 * 5.5) = 22.0$	Sim (4 para 4.0)
5 <sup>a</sup>	$3^a / 4^a$	$7.5 / 22.0 = 0.341$	Não
6 <sup>a</sup>	$v1 = 5^a$	$v1 = 0.341$	Não

# Conversão explícita de tipos

- É preciso **muito cuidado com a divisão inteira** (divisão entre operandos inteiros).
- O resultado da divisão inteira é sempre um número inteiro. Assim, se necessário, pode-se usar **uma conversão explícita de tipo** (*type casting*).

```
int a = 10, b = 3;
```

```
int c;
```

```
float d;
```

```
c = a / b;
```



c = 3

```
d = (float) a / b;
```



d = 3.333333

# Conversão explícita de tipos

- **Atenção!**

- Observe que os resultados de:

```
d = (float) a / b;
```

(1)

e

```
d = (float) (a / b);
```

(2)

são totalmente diferentes!

- Em (1), primeiro realiza-se primeiro a conversão explícita de tipo (**a** torna-se 10.0) e, em seguida, realiza-se a divisão. Logo: **d = 3.333333**.
- Em (2), primeiro divide-se **a** por **b** e, em seguida, se faz a conversão explícita de tipo. Logo: **d = 3.0**.

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b;
```

- E algumas operações:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b;
```

- E algumas operações:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	20
$b = (1 - 4) / (2 - 5)$	

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b;
```

- E algumas operações:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	<b>20</b>
$b = (1 - 4) / (2 - 5)$	<b>1.0</b>
$a = 2.75 + 1.24$	

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b;
```

- E algumas operações:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	<b>20</b>
$b = (1 - 4) / (2 - 5)$	<b>1.0</b>
$a = 2.75 + 1.24$	<b>3</b>
$b = a / 2.0$	



# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b;
```

- E algumas operações:

Operação de atribuição	Valor armazenado
$a = (2 + 3) * 4$	<b>20</b>
$b = (1 - 4) / (2 - 5)$	<b>1.0</b>
$a = 2.75 + 1.24$	<b>3</b>
$b = a / 2.0$	<b>1.5</b>

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b = 100.0;
```

- E algumas operações

Operação de atribuição	Valor armazenado
$a = 19 \% 5$	
$a = b \% a$	
$a = 1234 \% b$	
$b = a \% 2$	

# Escrevendo um programa em C

- Exemplo:

- Considere as seguintes declarações:

```
int a;  
float b = 100.0;
```

- Neste caso, teremos:

Operação de atribuição	Valor armazenado
$a = 18 \% 5$	3
$a = b \% a$	<b>Erro compilação!</b>
$a = 1234 \% b$	<b>Erro compilação!</b>
$b = a \% 2$	1.0

# Escrevendo um programa em C

```
1 #include <stdio.h>
2
3 void main() {
4     float n;
5     n = 10.0;
6     n = n * 1.1;
7     n = n / 2;
8     int a = n;
9     printf("a = %d, n = %f", a, n);
10 }
```

Endereço	Variável	Conteúdo
8512		
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1 #include <stdio.h>
2
3 void main() {
4     float n;
5     n = 10.0;
6     n = n * 1.1;
7     n = n / 2;
8     int a = n;
9     printf("a = %d, n = %f", a, n);
10 }
```

Endereço	Variável	Conteúdo
8512	<b>n</b>	
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1 #include <stdio.h>
2
3 void main() {
4     float n;
5     n = 10.0;
6     n = n * 1.1;
7     n = n / 2;
8     int a = n;
9     printf("a = %d, n = %f", a, n);
10 }
```

Endereço	Variável	Conteúdo
8512	n	10
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1 #include <stdio.h>
2
3 void main() {
4     float n;
5     n = 10.0;
6     n = n * 1.1;
7     n = n / 2;
8     int a = n;
9     printf("a = %d, n = %f", a, n);
10 }
```

Endereço	Variável	Conteúdo
8512	n	10
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1 #include <stdio.h>
2
3 void main() {
4     float n;
5     n = 10.0;
6     n = n * 1.1;
7     n = n / 2;
8     int a = n;
9     printf("a = %d, n = %f", a, n);
10 }
```

Endereço	Variável	Conteúdo
8512	n	11
8513		
8514		
8515		
8516		
8517		
8518		
8519		



# Escrevendo um programa em C

```
1 #include <stdio.h>
2
3 void main() {
4     float n;
5     n = 10.0;
6     n = n * 1.1;
7     n = n / 2;
8     int a = n;
9     printf("a = %d, n = %f", a, n);
10 }
```

Endereço	Variável	Conteúdo
8512	n	5.5
8513		
8514		
8515		
8516		
8517		
8518		
8519		

# Escrevendo um programa em C

```
1 #include <stdio.h>
2
3 void main() {
4     float n;
5     n = 10.0;
6     n = n * 1.1;
7     n = n / 2;
8     int a = n;
9     printf("a = %d, n = %f", a, n);
10 }
```

Endereço	Variável	Conteúdo
8512	n	5.5
8513	a	5
8514		
8515		
8516		
8517		
8518		
8519		

# Exercício

- Uma conta poupança foi aberta com um depósito de R\$500,00, com rendimentos 1% de juros ao mês. No segundo mês, R\$200,00 reais foram depositados nessa conta poupança. No terceiro mês, R\$50,00 reais foram retirados da conta. Quanto haverá nessa conta no quarto mês?