Universidade Federal do Espírito Santo
Centro Norte do Espírito Santo - CEUNES

Linguagens de Programação

Parte 02

Prof. Francisco Santos, Dr.

2021/2

Valores e Tipos de Dados

Valores e Tipos de Dados: Conceituação

- **Valor**

  3 2.5 'a' “Zé” 0x1F 026

- **Tipo**

- {true, 25, 'b', “azul” } composição de tipos
- { true, false } corresponde a um tipo

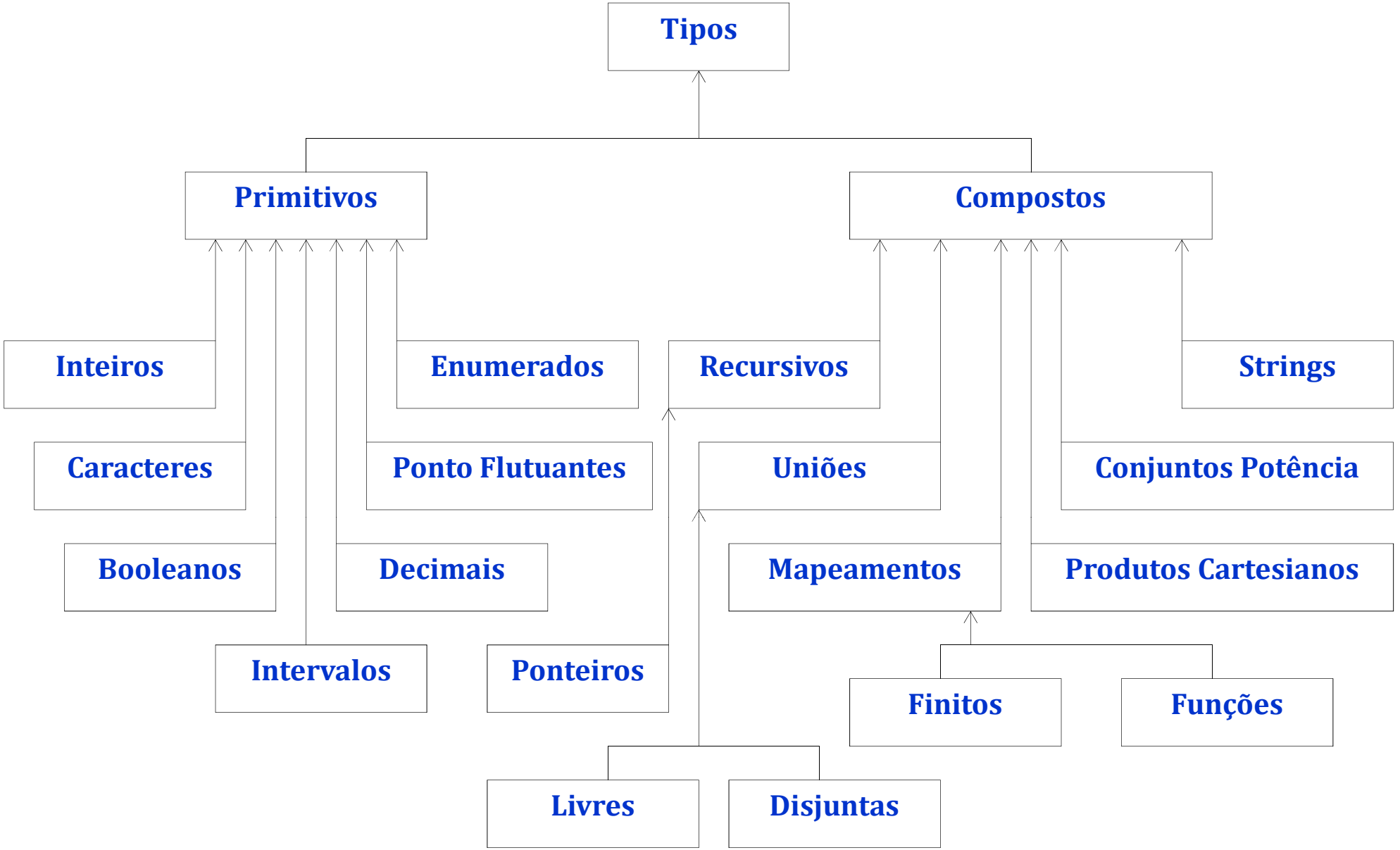
- Cardinalidade (#) de um tipo. Por exemplo, a cardinalidade do boolean é 2.

Valores e Tipos de Dados: Tipos Primitivos

- Não podem ser decompostos em valores mais simples
- Costumam ser definidos na implementação da LP

- Sofrem influência direta do hardware

Valores e Tipos de Dados: Hierarquia de Tipos



Tipos de Dados Primitivos:

Tipo Inteiro

- Corresponde a um intervalo do conjunto dos números inteiros
- Vários tipos inteiros numa mesma LP
  - Normalmente, intervalos são definidos na implementação do compilador
- Em JAVA, o intervalo de cada tipo inteiro é estabelecido na definição da própria LP

Tipos de Dados Primitivos:

Tipo Inteiro em Java

Tipo	Tamanho (bits)	Intervalo	
		Início	Fim
byte	8	-128	127
short	16	-32768	32767
int	32	-2.147.483.648	2.147.483.647
long	64	-9223372036854775808	9223372036854775807

Tipos de Dados Primitivos:

Tipo Inteiro em Java

```
System.out.println("Tipos de dados numéricos em Java: \n" +
  "\nMenor Byte: " + Byte.MIN_VALUE +
 "\nMaior Byte: " + Byte.MAX_VALUE +
   "\nMenor Short Int: " + Short.MIN_VALUE +
    "\nMaior Short Int: " + Short.MAX_VALUE +
     "\nMenor Int: " + Integer.MIN_VALUE +
     "\nMaior Int: " + Integer.MAX_VALUE +
      "\nMenor Long: " + Long.MIN_VALUE +
       "\nMaior Long:" + Long.MAX_VALUE +
        "\nMenor Float: " + Float.MIN_VALUE +
       "\nMaior Float: " + Float.MAX_VALUE +
        "\nMenor Double: " + Double.MIN_VALUE +
        "\nMaior Double: " + Double.MAX_VALUE);
```

Tipos de Dados Primitivos: Tipo Caractere

- Armazenados como códigos numéricos
- Tabelas ASCII e UNICODE
- PASCAL e MODULA 2 oferecem o tipo *char*
- Em C, o tipo primitivo *char* é classificado como um tipo inteiro

```
char d = 'a'+3;
printf("Valor = %d\n", d);
```

Diferente de:

```
char d[]= "a+3"; // NÃO É PRIMITIVO
printf("Valor = %s\n", d);
```

Tipos de Dados Primitivos:

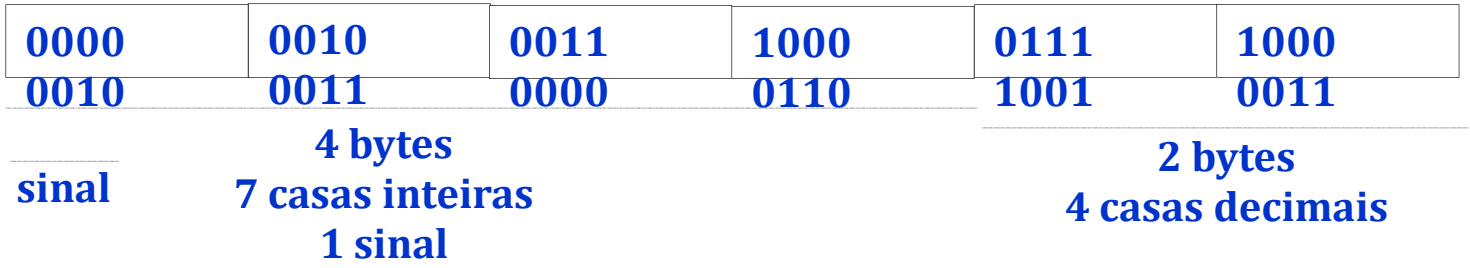
Tipo Booleano

- Tipo mais simples
  - Possui apenas dois valores
- C não possui o tipo de dado booleano, mas qualquer expressão numérica pode ser usada como condicional
  - Valores ≠ zero ⇒ verdadeiro
  - Valores = zero ⇒ falso
- JAVA inclui o tipo de dado *boolean*

Tipos de Dados Primitivos:

Tipo Decimal

- Armazena um número fixo de dígitos decimais
  - COBOL possui:



Tipos de Dados Primitivos:

Tipo Flutuante

- O tipo primitivo ponto flutuante modela os números reais
- LPs normalmente incluem dois tipos de ponto flutuante: *float* e *double*

Padrão IEEE 754 - Precisão Simples (float)

Padrão IEEE 754 - Precisão Dupla (double)

Tipos de Dados Primitivos:

Tipo Flutuante (Float X Double)

- O tipo primitivo ponto flutuante double possui o dobro de precisão do float, da seguinte maneira:

- double tem 52 bits mantissa + 1 bit oculto: $\log(2^{53}) \div \log(10) =$ 15,95 dígitos
- float tem 23 bits de mantissa + 1 bit oculto: $\log(2^{24}) \div \log(10) =$ 7,22 dígitos

Tipos de Dados Primitivos:

Tipo Flutuante (Float X Double)

- Vejamos um exemplo básico de cálculo em loop na Linguagem C utilizando float:

```
#include <stdio.h>
int main()
{
float a = 1.f / 81;
float b = 0;
for (int i = 0; i < 729; ++ i)
   b += a;
printf("%.7g\n", b); // prints 9.000023
}
```

Tipos de Dados Primitivos:

Tipo Flutuante (Float X Double)

- Agora, vejamos o memo exemplo utilizando double:

```
#include <stdio.h>
int main()
{
double a = 1.0 / 81;
double b = 0;
for (int i = 0; i < 729; ++ i)
  b += a;
printf("%.15g\n", b); // prints 8.99999999999996
}
```

Tipos de Dados Primitivos:

Tipo Flutuante *versus* Decimal

- Tipo **flutuante** é armazenado como uma parte significativa e um exponente, Por exemplo: 3,45.
- Nesse caso, o que é armazenado na memória é apenas 345 (a parte significativa) e -2 (o expoente ao qual a base 10 é elevada). Contudo, os computadores possuem a base binária.
- Números aparentemente simples precisariam de muitos dígitos para serem representados em binário. Precisariam de mais bits para serem representados com precisão. Então são arredondados. Muitas vezes isso não traz problemas, mas em alguns casos, como valores que passam por uma série de cálculos, sucessivos arredondamentos podem causar diferenças no resultado final.

Tipos de Dados Primitivos:

Tipo Flutuante *versus* Decimal

- Esse problema ocorre com os tipos float e double. O tipo double é como o tipo float, porém com uma maior precisão, portanto, é menos atingido por esse problema e consegue armazenar números maiores. Tipo Double utiliza mais bits para o armazenamento, também ocupa mais espaço em memória e em armazenamento de disco, no caso dos sistemas de banco de dados.

Tipos de Dados Primitivos:

Tipo Flutuante *versus* Decimal

- Por sua vez, o tipo **Decimal** não tem os mesmos problemas de precisão dos tipos float e double. Ele é um tipo de ponto fixo, e não flutuante. Por exemplo, se for definido um campo em um banco de dados como MySQL do tipo DECIMAL(10,2), quer dizer que será um número (em base 10) com 10 dígitos, sendo que 2 deles serão após a vírgula. Sempre.
- Essa estratégia de armazenamento elimina o problema dos arredondamentos no armazenamento dos dados, mas por outro lado isso diminui a flexibilidade do tipo. É uma boa opção para valores financeiros.

Tipos de Dados Primitivos:

Tipo Flutuante *versus* Decimal

- Porém, deve se levar em conta que em alguns casos, valores do tipo **decimal** serão convertidos em números de ponto flutuante de forma implícita quando forem usados em cálculos. Nesse caso se o resultado esperado também do tipo decimal, ele será arredondado e convertido de float/double para decimal.

Tipos de Dados Primitivos:

Tipo Enumerado

- PASCAL, ADA, C, C++ e java permitem que o programador defina novos tipos primitivos através da enumeração de identificadores dos valores do novo tipo, exemplo em java:

- public static enum mes_letivo {mar, abr, mai, jun, ago, set, out, nov};

- Possuem correspondência direta com intervalos de tipos inteiros e podem ser usados para indexar vetores
- **Aumentam a legibilidade e confiabilidade do código**

Tipos de Dados Primitivos:

Tipo Intervalo de Inteiros

- Em PASCAL e ADA, também é possível definir tipos intervalo de inteiros
- ```
  type meses = 1 .. 12;
  ```
- Tipos intervalos herdam as operações dos inteiros
  ```

Tipos de Dados Primitivos: Exercício

1) O número irracional π apesar de não ser possível ter um fração capaz de ser assumida como sua geratriz possui diversas aproximações de frações, por exemplo, 355/113.

As linguagens C/C++ possuem a constante M_PI da biblioteca math.h que introduz o número π com diversas casas decimais. Na linguagem Java é possível obter a referida constante por meio do comando Math.PI.

Agora, utilizando a constante π em uma das três linguagens citadas calcule a diferença absoluta entre a aproximação pela fração 355/113 e a constante obtida pelo comando nativo da linguagem escolhida. Para isso, realize dois cálculos: No primeiro armazene a constante e a fração em variáveis/atributos float, e realize a diferença entre a variável com que possui a fração com a variável que possui a constante π; Para ao segundo cálculo armazene os elementos em variáveis/atributos double e realize a mesma diferença do primeiro cálculo. Por fim, identificar o cálculo que obteve a menor diferença em relação a constante π. Por que isso ocorre? E se forem impressos na tela as variáveis float e double que armazenam os valores da fração que aproxima de π, o que é possível perceber?

Tipos de Dados Primitivos: Exercício

2) Crie um novo tipo de dados em C/C++/Java chamado QUARTERNARIO, podendo assumir: FALSO, VERDADEIRO, INTERMADIARIO, MAXIMO.

Crie outras variáveis e atribua os estados de QUARTERNARIO. Realize operações de comparação. Por exemplo, leia cinco notas de alunos, calcule a média, declare uma variável status_aprovacao, atribua VERDADEIRO se nota maior que sete, FALSO se menor que sete, INTERMEDIARIO se estiver entre cinco e sete, e MAXIMO se nota maior que nove. Posteriormente, exiba mensagens parabenizando ou sugerindo estudo ao aluno dependendo do seu status_aprovacao.

Tipos de Dados Primitivos: Exercício

```
public class TiposVariaveisENUM {

   public static enum QUARTERNARIO {  VERDADEIRO, FALSO, INTERMEDIARIO, MAXIMO }

   public static QUARTERNARIO valor;

 public static void main(String[] args) {
       float media;
       Scanner entrada = new Scanner(System.in);

       System.out.print("Digite a média: ");
       media = entrada.nextFloat();

     if (media>=7)
       valor = QUARTERNARIO.VERDADEIRO;
```

Tipos de Dados Primitivos: Exercício

```
else if (media<7)
            valor = QUARTERNARIO.FALSO;

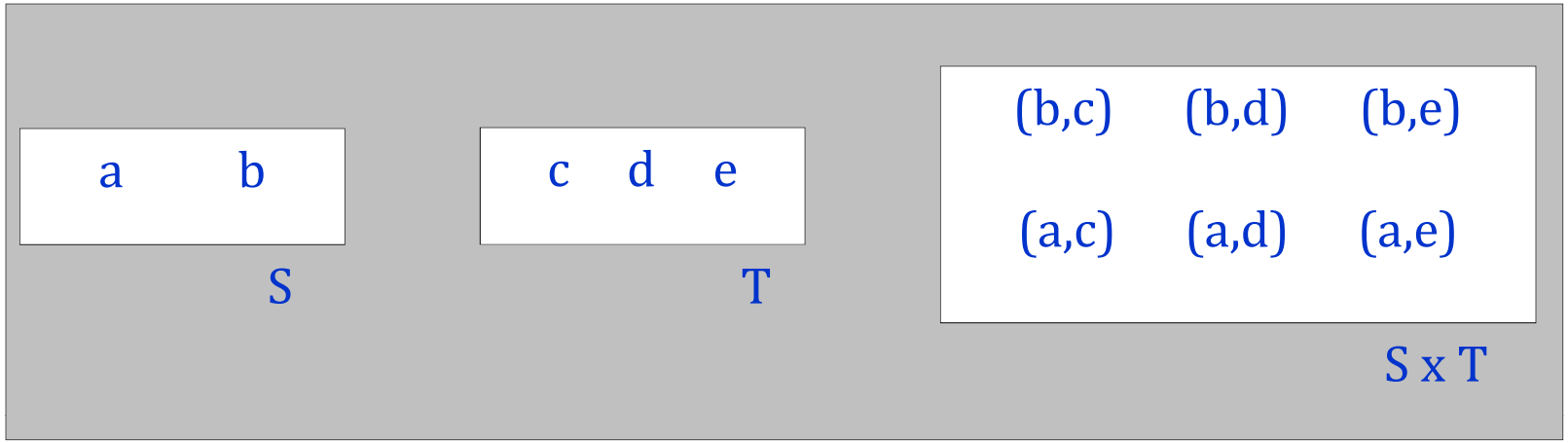
      if (valor.equals(QUARTERNARIO.VERDADEIRO))
        System.out.print("O(A) aluno(a) está de parabéns, foi aprovado(a)! ");
       else if (valor.equals(QUARTERNARIO.FALSO))
          System.out.print("O(A) aluno(a) foi Reprovado! ");
   }
```

Tipos de Dados Compostos

- Tipos compostos são aqueles que podem ser criados a partir de tipos mais simples (primitivos), São exemplos:
  - registros, vetores, listas, arquivos
- Entendidos em termos dos conceitos:
  - **produto cartesiano**, **uniões**, **mapeamentos**, **conjuntos potência** e **tipos recursivos**
- Cardinalidade
  - número de valores distintos que fazem parte do tipo

Tipos de Dados Compostos: Produto Cartesiano

- Combinação de valores de tipos diferentes em tuplas



Tipos de Dados Compostos: Produto Cartesiano

- São produtos cartesiano os registros de PASCAL, MODULA 2, ADA e COBOL e as estruturas de C.

```
struct tipoNome {
char primeiro [20];
char meio [10];
char sobrenome [20];
}
struct empregado {
struct tipoNome nomeFuncionario;
float salario;
} empregado;
```

- **Uso de seletores**
  - empregado.nomeFuncionario.sobrenome

Tipos de Dados Compostos: Produto Cartesiano

- Inicialização em C

  struct data { int d, m, a; };

  struct data d = { 7, 9, 1999 };

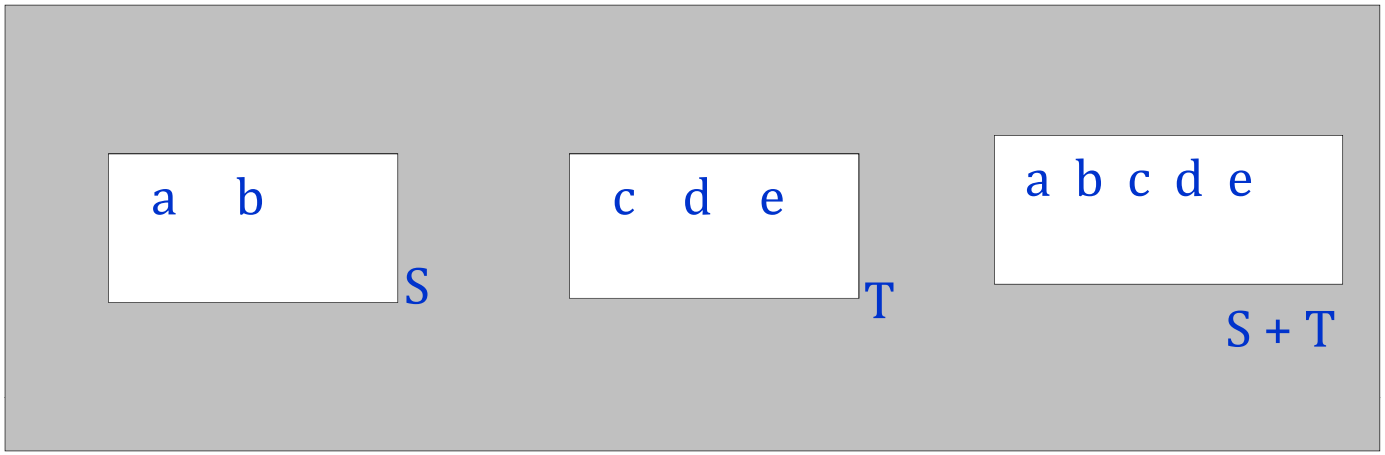
- Em LPs orientadas a objetos, produtos cartesianos são definidos a partir do conceito de classe
  - JAVA só tem class

- Cardinalidade

  #(S1 x S2 x ... x Sn) = #S1x #S2x ...x #Sn

Tipos de Dados Compostos: Uniões

- Consiste na união de valores de tipos distintos para formar um novo tipo de dados



Tipos de Dados Compostos: Uniões

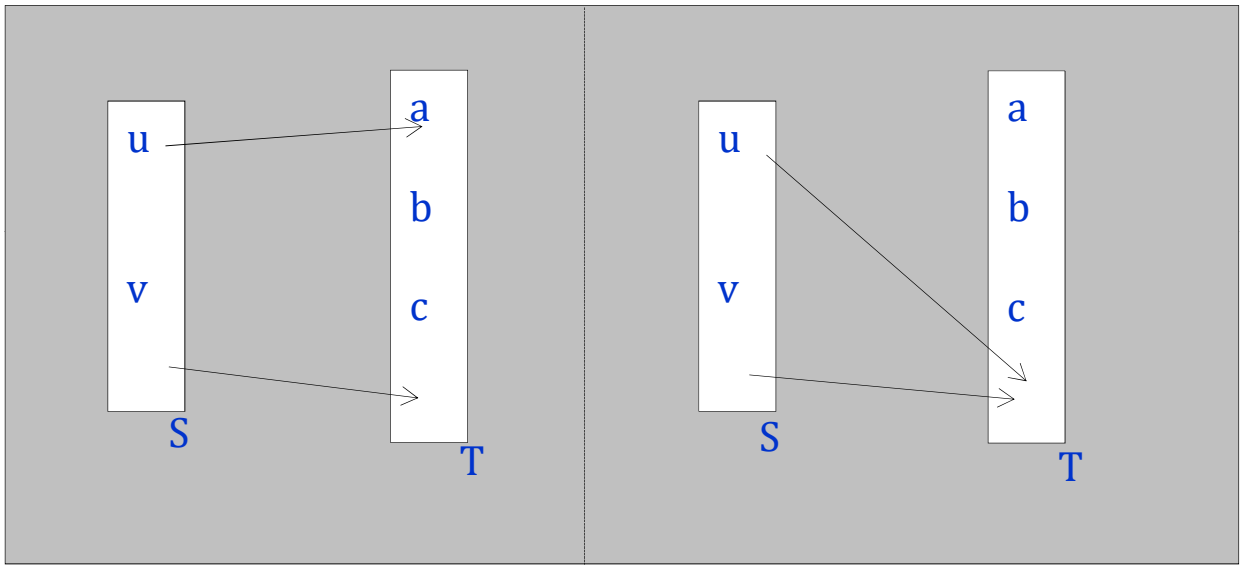
- Uniões Livres
  - Pode haver interseção entre o conjunto de valores dos tipos que formam a união

```
union medida {
  int centimetros;
  float metros;
  };
```

```
union medida medicao;
medicao.centimetros=180;
```

Tipos de Dados Compostos: Mapeamentos

- Tipos de dados cujo conjunto de valores corresponde a todos os possíveis mapeamentos de um tipo de dados S em outro T

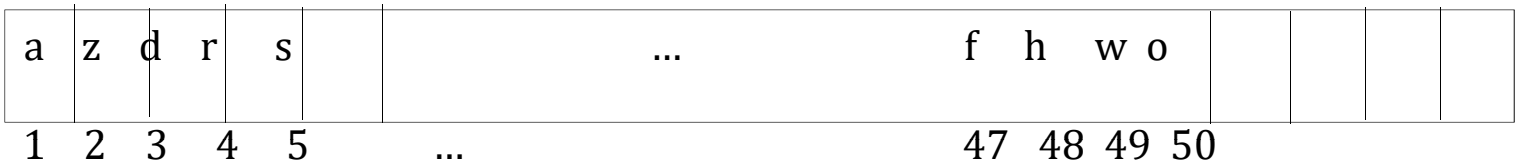


Tipos de Dados Compostos: Mapeamentos Finitos

- O conjunto domínio é finito
- Vetores e matrizes

array S OF T; (S → T)

A: array [1..50] of char; A: ([1,50] → char)



- O conjunto índice deve ser finito e discreto
- Verificação de limite dos índices (Pascal, JAVA)

```
int v[7];       //em C
V[13] = 198;
int[] vet = new int[2147483647]; // Em Java
```

Tipos de Dados Compostos:

Maps em C++ e Java

- Recurso útil para buscas diretas apenas com um elemento chave
- Possui métodos pré definidos nas interfaces
- Utiliza iterator em C++
- C++ e Java possuem uma ampla gama de métodos para inserções, buscas, listagens, alterações, remoções, entre outros
- Há gerenciamento do tamanho do Map. Em Java todo Map quando definido possui por padrão tamanho 16. Um flag chamado load factor = 0.75, é utilizado para "identificar" quando está próximo do tamanho máximo atual do Map e assim aumentar seu tamanho. Isso ocorre a medida que estão sendo inseridos os dados.
- Por exemplo, em Java, quando chega-se aos 12 primeiros elementos inseridos (16*0.75 = 12) internamente ocorre uma atualização e o Map dobra de tamanho.

Tipos de Dados Compostos:

Exemplo de Map em C++

```
// Definicao do mapeamento no formato de par: chave e valor
   map<int, string> Mapeamento_LP;

   // Insere os dados
   int CPF=3385; string Nome="Ana Paula";
   Mapeamento_LP.insert(pair<int, string>(CPF, Nome));

  // Imprime elementos
        cout << "\nO Mapa é : \n";
        cout << "\tChave\tNome\n";
        map<int, string>::iterator itr;
        for (itr = Mapeamento_LP.begin(); itr != Mapeamento_LP.end(); ++itr) {
         cout << '\t' << itr->first
         << '\t' << itr->second
         << '\n';
        }    cout << endl;

   // Busca um elemento específico por meio da chave associada ao referido elemento
   int busca = 3385;
   cout << "Valor da Chave Buscada => " << Mapeamento_LP.find(busca)->second << '\n';
```

Tipos de Dados Compostos:

Exemplo de Map em Java

```
// Definicao do mapeamento no formato par: chave e valor
Map<Integer, String> Mapeamentos_LP = new HashMap<Integer, String>();

// Inserção de elementos
Mapeamentos_LP.put(4485, new String("Francisco"));
Mapeamentos_LP.put(3344, new String("Ana Paula"));


//Impressão dos elementos
 System.out.println("Chave   Valor ");
          for (Integer key : Mapeamentos_LP.keySet()) {

               //Capturamos o valor a partir da chave
               String value = Mapeamentos_LP.get(key);
               System.out.println(key + "    " + value);
          }
 // Busca direta  utilizando a chave
      Integer chave = 3344;
      String Nome = Mapeamentos_LP.get(chave);
      System.out.println("\nO valor da chave buscada eh: " + Nome);
```

Tipos de Dados Compostos: Exercícios

3) Desenvolver uma rotina em C++/Java que utilize Map para inserir um conjunto de informações de várias pessoas, incluindo Nome, CPF, Data de Nascimento e Profissão. A chave deve ser o CPF. Para isso recomenda-se definir uma classe chamada Pessoas com seus respectivos atributos. Também, pode ser adequado fazer uso de switch-case para apresentar opções de Inserir dados, Pesquisar uma chave e obter os dados associados, Alterar dados, Remover registros, Quantidade de registros do Map, Imprimir as informações inseridas, e sair. A solução deve ser modularizada, construída com métodos para inserir dados, imprimir e busca direta por uma chave bem como os demais métodos.

Categorias de Vetores

- Estáticos (C)

```
void f () {
     static int x[10];
}
int count (void) {
   static int num = 0;
   num++;
   return num;
 }
```

ou:

```
int x[10];
void f () {
    ...
}
```

- Semi-Estáticos (C)

```
void f () {
     int x[10];
}
```

- Semidinâmicos

```
char * p;
p=(char*)malloc(sizeof(char));
```

ou:

```
void f (int n) {
    int x[n];
}
```

- Dinâmicos (APL)

```
A ← (2  3  4)
A ← (2  3  4  15 )
```

Vetores Dinâmicos

- Podem ser implementados em C, C++ e JAVA através do monte (heap) - diferente de *stack* (pilha). No heap utiliza-se qualquer área disponível.
- Necessário alocar nova memória e copiar conteúdo quando vetor aumenta de tamanho
- É encargo do programador controlar alocação e cópia. Em C e C++, o programador deve controlar desalocação também. Isso torna a programação mais complexa e suscetível a erros

Vetores Dinâmicos

- Podem ser implementados JAVA:
  - ArrayList<String> lista = new ArrayList<String>();
    lista.add("Abacaxi");
    lista.add("Laranja");
    lista.add("Pera");

/* Quantos elementos quiser, desde que caiba na memória */ for (String s : lista)

  - System.out.println(s);

OBS: Internamente ocorre a redefinição de um novo array maior e cópia do conteúdo (do atual) para o novo vetor. Ao contrário do C e C++ o java gerencia tudo isso.

Vetores Multidimensionais

- Em JAVA vetores multidimensionais são vetores unidimensionais cujos elementos são outros vetores

```
int [] [] a = new int [5] [];
for (int i = 0; i < a.length; i++) {
    a [i] = new int [i + 1]; //cria duas posições em cada índice i
}
```

- O mesmo efeito pode ser obtido em C com o uso de ponteiros para ponteiros

Vetores Multidimensionais

- Cardinalidade

  int mat [5][4];

  Conjunto de valores

  $\{0, \ldots, 4\} \times \{0, \ldots, 3\} \rightarrow int$

  $(\#int)^{\#(\{0, \ldots, 4\} \times \{0, .., 3\})} =$

  $(\#int)^{(\#\{0, \ldots, 4\} \times \#\{0, .., 3\})} =$

  $(\#int)^{5 \times 4} =$

  $(\#int)^{20}$

Mapeamentos Através de Funções

- Uma função implementa um mapeamento S→T através de um algoritmo
- O conjunto S não necessita ser finito
- O conjunto de valores do tipo mapeamento S→T são todas as funções que mapeiam o conjunto S no conjunto T
- Valores do mapeamento [int → boolean] em JAVA

```
boolean  positivo (int n) {
        return n > 0;
}
```

Mapeamentos Através de Funções

- C utiliza o conceito de ponteiros para manipular endereços de funções como valores

```
int impar (int n){ return n%2; }
int negativo (int n) { return n < 0; }
int multiplo7 (int n) { return !(n%7); }
int conta (int x[], int n, int (*p) (int) ) {
    int j, s = 0;
    for (j = 0; j < n; j++)
        if ( (*p) (x[j]) ) s++;
    return s;
}
main() {
    int vet [10] = {1, 5, 8, -7, 124, 2, 8, 6, 21, -9,};
    printf ("%d\n", conta (vet, 10, impar));
    printf ("%d\n", conta (vet, 10, negativo));
    printf ("%d\n", conta (vet, 10, multiplo7));
 }
```

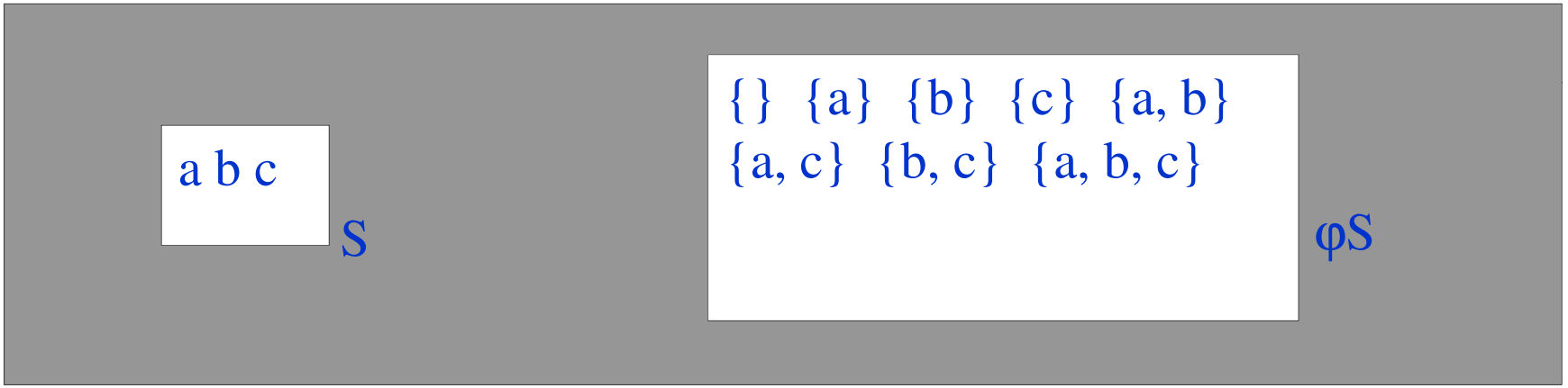
Mapeamentos Através de Funções

- JAVA não trata funções (métodos) como valores
- Pode-se empregar algoritmos diferentes para implementar um mesmo mapeamento
- Algumas vezes, vetores e funções podem ser usados para implementar o mesmo mapeamento finito

Conjuntos Potência

- Tipos de dados cujo conjunto de valores corresponde a todos os possíveis subconjuntos que podem ser definidos a partir de um tipo base S

$\varphi S = \{s \mid s \subseteq S\}$



Conjuntos Potências

- Cardinalidade

  $\#\varphi S = 2^{\#S}$

- Operações básicas

  Pertinência

  Contém

  Está contido

  União

  Diferença

  Diferença simétrica

  Interseção

Conjuntos Potências

- Poucas LPs oferecem. Exemplo em Pascal:

```
TYPE
     Carros = (corsa, palio, gol);
     ConjuntoCarros = SET OF Carros;
 VAR
     Carro: Carros;
     CarrosPequenos: ConjuntoCarros;
  BEGIN
     Carro:= corsa;
     CarrosPequenos := [palio, gol];                  /*atribuicao*/
     CarrosPequenos:= CarrosPequenos + [corsa];  /*uniao*/
     CarrosPequenos:= CarrosPequenos * [gol];      /*intersecao*/
     if Carro in CarrosPequenos THEN                 /*pertinencia*/
     if  CarrosPequenos >= [gol, corsa] THEN      /*contem*/
```

Tipos Recursivos

- Tipos recursivos podem ser definidos a partir de ponteiros ou diretamente

Em C

```
struct no {
   int  elem;
   struct no* prox;
};
```

Em C++

```
class no {
   int elem;
   no* prox;
};
```

Em JAVA

```
class no {
   int elem;
   no prox;
};
```

Tipos Ponteiros

- Não se restringe a implementação de tipos recursivos embora seja seu uso principal
- Ponteiro é um conceito de baixo nível relacionado com a arquitetura dos computadores
- O conjunto de valores de um tipo ponteiro são os endereços de memória e o valor nulo (nil, null)
- Considerados o goto das estruturas de dados

Tipos Ponteiros

```
#define nil 0
typedef struct no* listaInt;
struct no {
   int cabeca;
   listaInt cauda;
};
listaInt anexa (int cab, listaint caud) {
   listaInt l;
   l = (listaInt) malloc (sizeof (struct no));
   l->cabeca = cab;
   l->cauda = caud;
   return l;
}
```

Tipo Referência

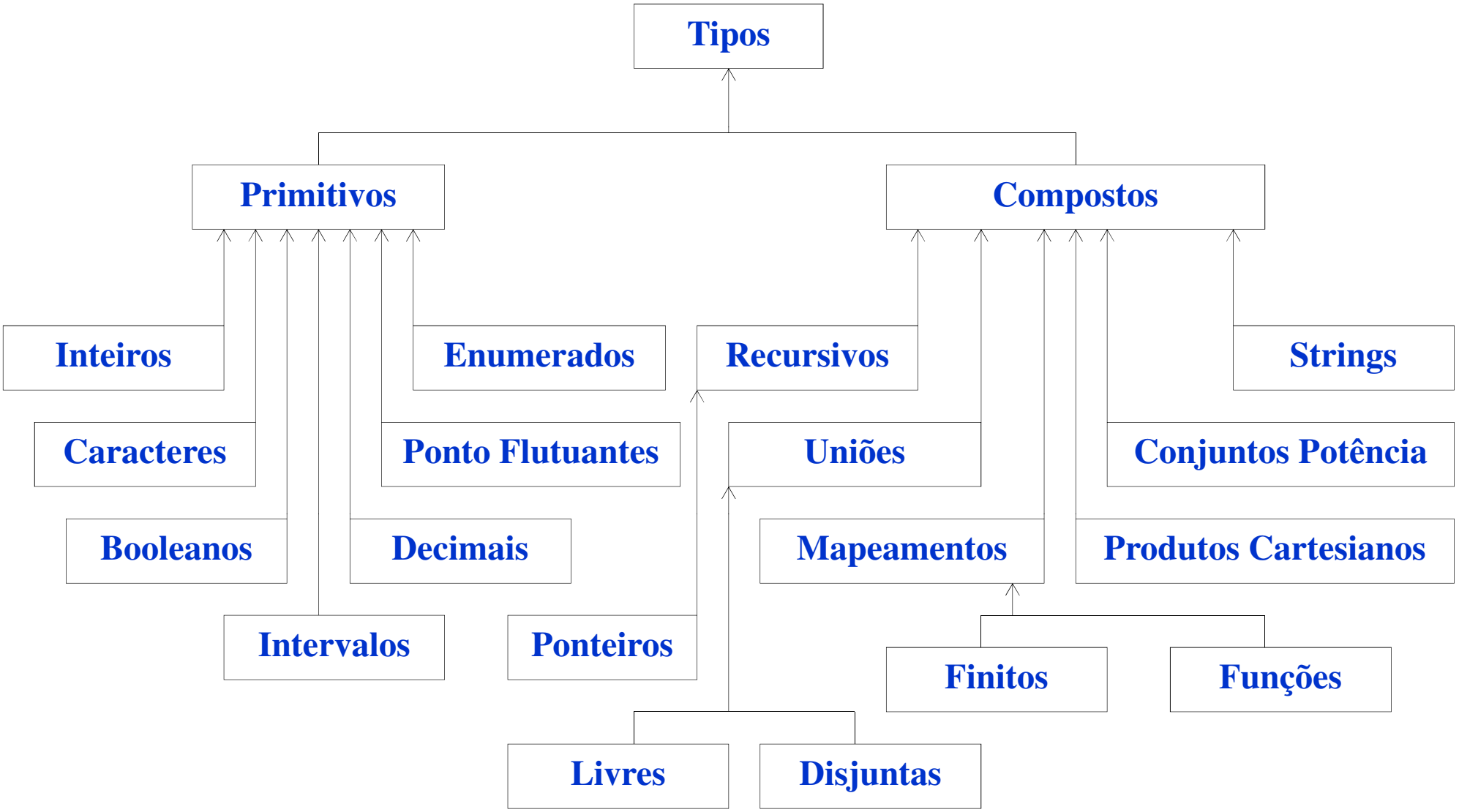
- O conjunto de valores desse tipo é formado pelos endereços das células de memória
- Todas as variáveis que não são de tipos primitivos em JAVA são do tipo referência
- Em C++

```
int valor = 0;
int& ref = valor;              // 'ref' passa a referenciar 'valor'
ref = 100;              // 'valor' passa a valer 100
```

Tipo String

- Valores correspondem a uma seqüência de caracteres
- Não existe consenso sobre como devem ser tratadas
- Podem ser consideradas tipos primitivos (PERL, SNOBOL, ML), mapeamentos finitos (C, PASCAL) ou tipo recursivo lista (PROLOG, LISP)
- Três formas comuns de implementação
- Estática (em COBOL)
- Semi-Estática (em C, PASCAL)
- Dinâmica (em PERL, SNOBOL, APL)

Hierarquia de Tipos



Atividades

Implemente uma rotina em qualquer linguagem de programação, permitindo:

1) Realizar operações de soma, subtração, multiplicação e divisão (de dois valores)
2) Realizar testes (primo, par e positivo) dos N primeiros números negativos até os N números positivos.

Para isso aplique os conceitos de Mapeamentos Finitos em particular, Mapeamentos através de Funções. Disponibilizar menus de opções, Sendo:

1 – Operações aritméticas

2 – Realizar testes

3 – Sair

Na opção 1 deve apresentar outro menu de opções para cada uma das operações, o memo para a opção 2, com as alternativas de testes.

Atividades

Exemplo da primeira rotina:

```
#include <stdio.h>
float soma(float num1, float num2){    return (num1+num2);}
float divisao(float num1, float num2){    return (num1/num2);}
float calculadora (float (*p) (float, float) ) {
   float num1,num2;
   printf("Digite o primeiro valor\n");
   scanf("%f",&num1);
   printf("Digite o segundo valor\n");
   scanf("%f",&num2);
return ( (*p)(num1,num2) );}
main() {
printf ("%f\n", calculadora(soma));  /* Deve-se usar um case*/
printf ("%f\n", calculadora(divisao));}
```

Atividades

Estruture uma rotina em C/C++ para realizar o controle de vendas de ingressos para um grande evento com capacidade de 4000 lugares. Os dados dos ingressos são: Nome evento, Valor do Ingresso, Data da venda, Horário do Evento, Data do Evento e Nome do Participante. A rotina deve de contabilizar os ingresso vendidos. Para isso utilize os conceitos de **Produto Cartesiano e Mapeamentos Finitos** na terminologia de tipos de dados compostos.

Atividade 1 – Parte 3

Implemennte em C/C++/Java três funções para realizar a soma, subtração e a divisão entre dois números.

No programa principal solicitar ao usuário que leia dois valores quaisquer a e b, não simétricos. Se a > b então calcular a subtração entre esses dois valores, senão, calcular a divisão. Porém, independente do resultado da comparção realizar a soma Para isso utilize expressão composta sem curto circuíto.

Atividade 1 – Parte 3

```
public class Principal {

static float soma(float a, float b) {
float soma = a+b;
System.out.println("soma: "+soma);
return soma;
}
static float divisao(float a, float b) {
float divisao=a/b;
System.out.println("divisao: "+divisao);
return divisao;
}
```

Atividade 1 – Parte 3

```
static float subtracao(float a, float b) {
float subtracao=a-b;
System.out.println("subtração: "+subtracao);
return subtracao;
}
public static void main(String[] args) {
int a = 9, b = 33, c = 4;
if( (a > b) & (soma(a,b)!=0)  )
  subtracao(a,b);
if ( a < b )
  divisao(a,b);
 }
}
```