Centro Tecnológico
Departamento de Informática

Prof. Veruska Zamborlini
veruska.zamborlini@inf.ufes.br
http://www.inf.ufes.br/~veruska.zamborlini

Aula 5/6
Tipos de Dados
2021/2

Material adaptado
Prof.s Vitor Souza e Eduardo Zambon

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

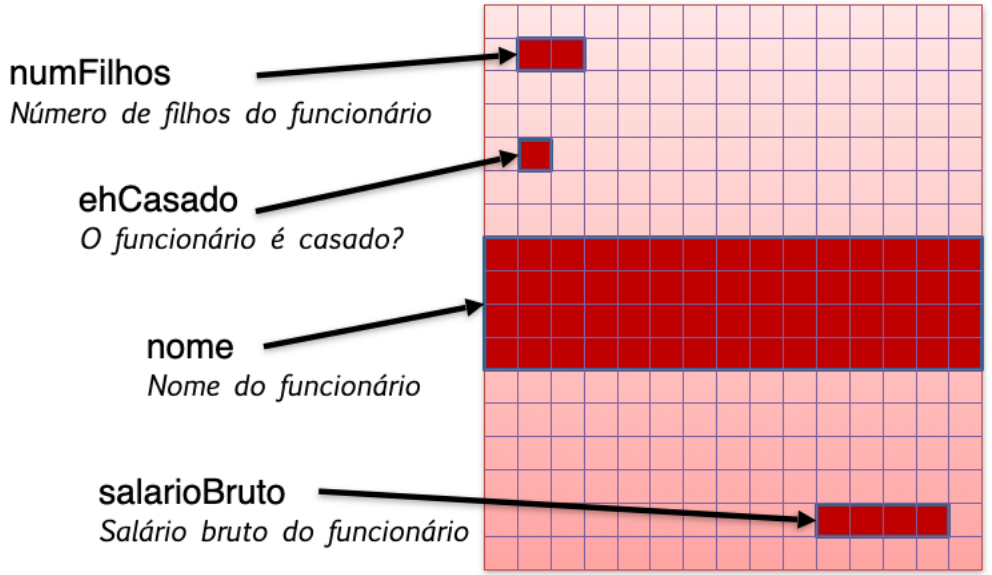
Tipos de Dados

- Um tipo de dados define um conjunto de valores e uma coleção de operações sobre estes.

- Exemplo:
  - O tipo inteiro ***int*** em C compreende*:
    - Números de -32.768 a +32.767
    - Operações: soma(+), subtração(-), multiplicação(*), divisão inteira(/), resto da divisao(%), incremento(++), decremento(--)

Tipos de Dados

- Dá significado a uma sequência de bits armazenados na memória.



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos de Dados

- Possuem cardinalidade: número de valores distintos que fazem parte do tipo;
  - Em geral, possuem número fixo de valores;
    - Há exceções: fracionários em SmallTalk, Integer de Haskell, BigInteger em Java;

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos de Dados

- Estabelece a qtde de memória necessária para armazenar um valor -> limita a qtde de valores possíveis.

- Exemplo para um compilador C em hardware com palavra* de 16 bits:

*unidade de transferência entre a CPU e memória principal

Tipo	Num de bits	Intervalo	
		Inicio	Fim
char	8	-128	127
unsigned char	8	0	255
signed char	8	-128	127
int	16	-32.768	32.767
unsigned int	16	0	65.535
signed int	16	-32.768	32.767
short int	16	-32.768	32.767
unsigned short int	16	0	65.535
signed short int	16	-32.768	32.767
long int	32	-2.147.483.648	2.147.483.647
signed long int	32	-2.147.483.648	2.147.483.647
unsigned long int	32	0	4.294.967.295
float	32	3,4E-38	3.4E+38
double	64	1,7E-308	1,7E+308
long double	80	3,4E-4932	3,4E+4932

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Sistema de Tipos & LP's

- Conjunto de tipos e as regras que governam seu uso em linguagens de programação.

- Junto com as estruturas de controle, eles formam o "coração" de uma linguagem.

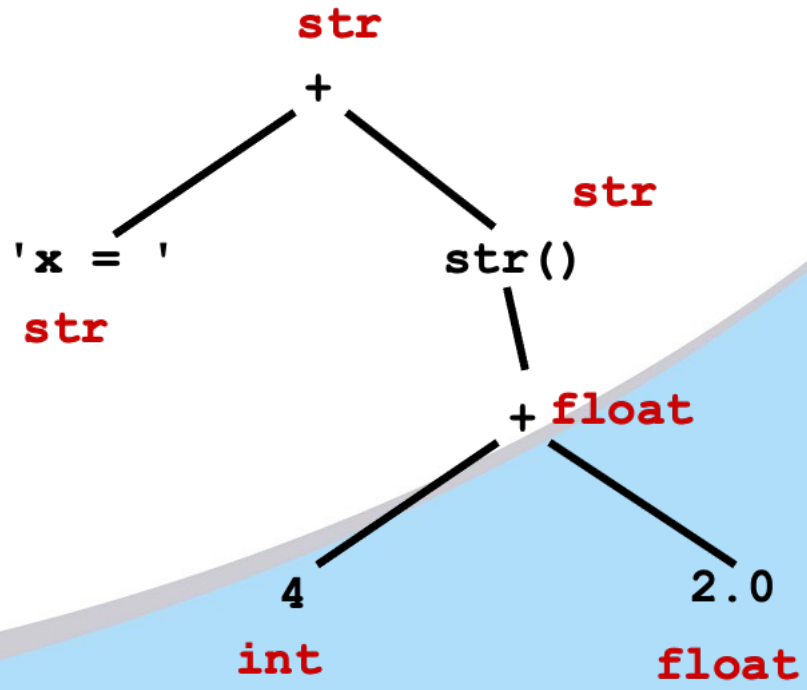
- Exemplo:
  - O tipo string faz parte do sistema de tipos de Python, mas não de C. Consequentemente, é muito mais "fácil" manipular strings em Python do que em C.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Sistema de Tipos & LP's

- Associa um tipo a cada expressão da LP
- Inclui regras para:
  - Equivalência de tipos
  - Compatibilidade de tipos

```
'x = ' + str(4 + 2.0)
```



O que acontece se '+' não representa uma operação pré-definida para o tipo str?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Sistema de Tipos & LP's

- Um ***sistema de tipos*** provê diversos benefícios para uma LP.
  - Detecção de erros
  - Assistência para modularização
  - Documentação

- Linguagens de baixo nível não tem sistema de tipos **=> ?**

Não há diferentes tipos de "significados" para sequências de bits, nem operações específicas, tampouco detecção de erro.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Sistema de Tipos & LP's

- Uma LP pode ser ***fortemente tipada*** se:
  - "***Sempre***" detecta erros de tipo
  - … ou …
  - Torna suficientemente difícil subverter o sistema de tipos.
  - **Definição muda entre autores**

- Grau de força de uma LP:
  - Quão difícil é para subverter o sistema de tipos?
    - C/C++ são fracamente tipadas (*weakly typed*) - e.g. unions
    - ML e Haskell são fortemente tipadas (*strongly typed)*
    - Java é "quase fortemente tipada" (*nearly strongly typed*)

Tipagem / Vinculação de tipos

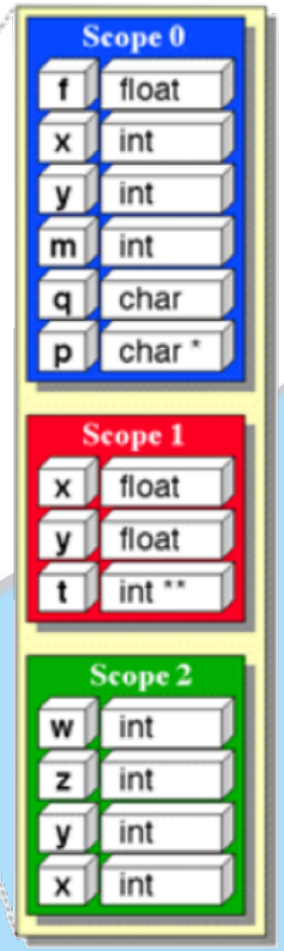
- Estática - antes da execução / não muda
  - Java, C, Pascal, Fortran, etc.;
- Dinâmica - durante execução / muda
  - Perl, Python, Scheme, etc.

```
float   f;
int     x, y, m;
char    q, *p;
. . .
{
        float   x, y;
        int     **t;
        . . .
        {
                int     w, z, y, x;
                . . .
```

Symbol Table

scopes
symbols
attributes

Create
AddSymbol
GetAttributes
OpenScope
CloseScope
Destroy



Detecção de erros no tempo de vinculação!

Tipos de Tipos

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos de Tipos

- Tipos podem representar valores simples (primitivos) ou compostos:
  - Caracter ‘a’ ou sequência ‘abcd’

- Decisões de projeto de LPs podem determinar quais são os tipos são implementados e se primitivos e compostos.

- Importante conhecer para cada LP que se pretende aprender quais tipos são suportados e como.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos Primitivos

- Não podem ser decompostos em valores mais simples;
- Não é definido em termos de outros tipos!

- Exemplos:
  - Tipos numéricos
  - Tipos booleanos
  - Caracteres
  - ? (varia com o projeto da linguagem)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos Compostos

- Criados a partir de tipos simples/primitivos.

- Veremos a seguir teoria sobre formas de combinar os tipos existentes, por exemplo, produto cartesiano ou mapeamento.

- Exemplos:
  - Vetores (mapeamento)
  - Matrizes (mapeamento)
  - Registros (produto cartesiano)
  - ? (varia com o projeto da linguagem)

Tipos Primitivos

(possivelmente)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Caracteres

- Armazenados como códigos numéricos:
  - Tabelas EBCDIC, ASCII e UNICODE;
  - ASCII < Latin1 ISO 8859-1 < UTF-8 < UTF-16 < UTF-32;
  - Pode causar problemas, por exemplo, na leitura de arquivos

- Pascal e Modula-2 oferecem o tipo char;
- Em C, char é classificado como um tipo inteiro;
- Em Python, um caracter é uma string de tamanho 1.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Cadeias de Caracteres (string)

- Tipo primitivo ou tipo especial de vetor de caracteres?

- Em C string não é um tipo primitivo
  - A princípio é simplesmente um vetor
  - Porém pode-se usar uma biblioteca padrão para se ter acesso a operações específicas para “strings”

- Em Python string é um tipo primitivo

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Cadeias de Caracteres (string)

- Qual o limite de tamanho?
  - Tamanho estático (static length string)
  - Tamanho dinâmico limitado (limited dynamic length)
  - Tamanho dinâmico (dynamic length string)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Cadeias de Caracteres (string)

- ***Tamanho estático (static length string)***
- Strings são imutáveis, tamanho e valor.
- Exemplo, Python

```
$ python
>>> s = "abc"
>>> len(s)
3
>>> s.append('d')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'str' object has no attribute 'append'
```

- Entretanto, a variável s pode “aumentar”, apontando pra outro objeto “abcd”:

```
$ python
>>> s += "d"
>>> len(s)
4
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Cadeias de Caracteres (string)

- ***Tamanho dinâmico limitado (limited dynamic length)***
- Há um tamanho máximo porém a string pode variar o tamanho da string pode variar, desde que seja menor que o máximo.
- Exemplo, C

```
#include <iostream>
using namespace std;

int main() {
    char a[100] = {'O', 'l', 'a'};
    cout << a << '\n';           // Ola
    cout << sizeof a << '\n';    // 100
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Cadeias de Caracteres (string)

- ***Tamanho dinâmico (dynamic length string)***
- Permite strings com tamanho variável, sem um máximo
- Exemplo: Perl

```
#!/usr/bin/perl -l

$s = "abc";
print length($s); # 3
$s .= "d";
print length($s); # 4
```

- Perl usa realloc, que pode ou não requerer um novo buffer.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Booleanos

- Tipo mais simples: possui apenas dois valores, verdadeiro ou falso;
- C e Perl não possuem booleano, mas qualquer expressão numérica pode ser usada como condicional:
  - ≠ zero: verdadeiro; = zero: falso;
- Java inclui o tipo de dado boolean;
- Ocupa geralmente 1 byte (difícil endereçar 1 bit);

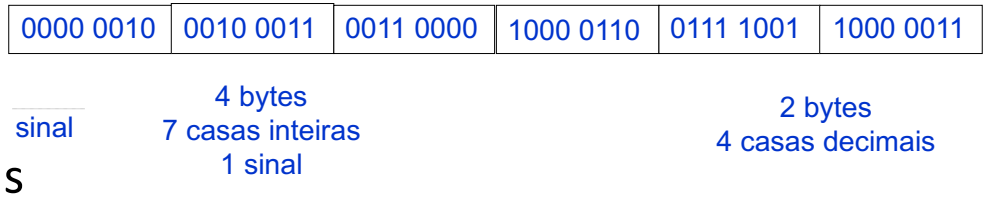
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Inteiros

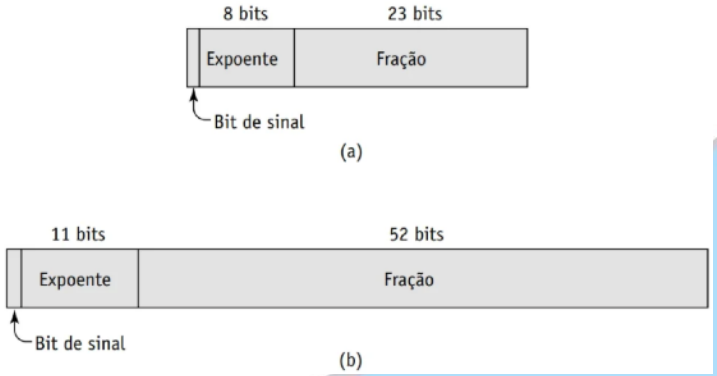
- Corresponde a um intervalo do conjunto dos números inteiros;
- Uma mesma LP pode ter vários intervalos;
  - C possui char e int, com modificadores signed, unsigned, short e long.
- Podem ser definidos:
  - na implementação do compilador: C
    - O tipo int possui o tamanho da palavra da arquitetura em questão.
  - na definição da LP: Java
  - pelo programador: Ada (dá erro caso exceda os limites)

Reais / Complexos

- Decimal: (e.g. COBOL)
  - Armazena um número fixo de dígitos decimais
  - Vantagem: precisão
  - Desvantagem: desperdício de memória, expressividade reduzida, operações complexas



- Ponto flutuante: (e.g. C)
  - Modela os números reais em expoente e fração;
  - LPs normalmente incluem dois tipos: float e double;
  - Imprecisão: 3.231 * 100 = 323.100006 (float em C).



- Complexo: (e.g. Python e Fortran 77)
  - Python: par ordenado para representar as partes real e imaginária.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos Ordinais definidos pelo usuário

- Enumeração

```
enum days {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

  - Geralmente são convertidos para inteiros
  - Melhora a legibilidade, mas pode comprometer a confiabilidade se operações de tipo não forem limitadas.
    - Eg. x = Mon + Fri

Tipos Compostos

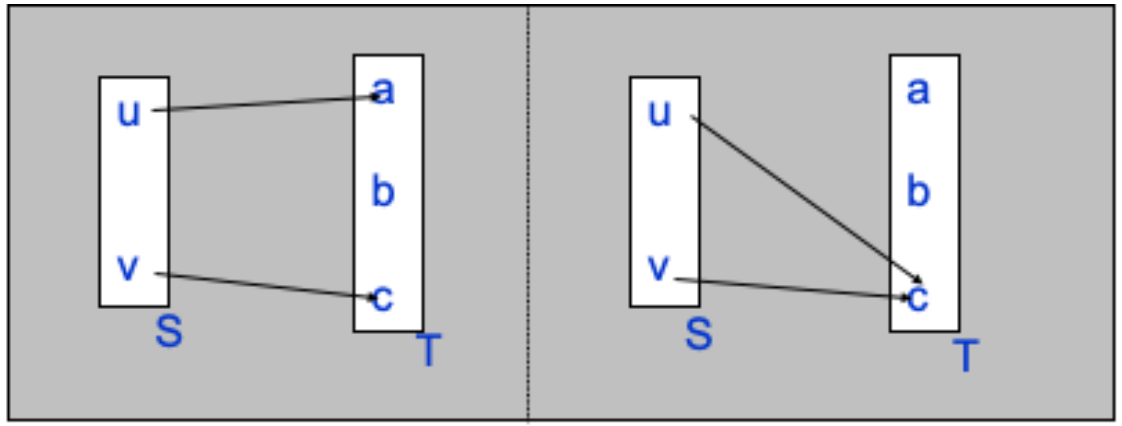
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos Compostos

- Criados a partir de tipos simples/primitivos
- Entendidos em termos dos seguintes conceitos (Watt 1990):
  - Produto cartesiano; struct C
  - Uniões; union C
  - Mapeamentos; arrays, dicionários ou funções
  - Conjuntos potência; poucas linguagens
  - Tipos recursivos. nó aponta pra outro nó

Mapeamentos

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



- Funciona como função injetora (todos de S estão incluídos, mas não necessariamente todos de T);

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Mapeamentos finitos

- O conjunto domínio é finito;
- Exemplos:
  - Arranjos/*Arrays*: Vetores e matrizes;
  - Dicionários (em Python);
  - Acesso a valores via índice:
    - O conjunto domínio (índices) deve ser finito e discreto;
    - Em muitas linguagens informa-se índice entre colchetes, algumas porém entre parênteses

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

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;
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Arrays (vetores/matrizes)

- Escolhas de projeto/implementação:
  - Inicialização na declaração?
  - Valores permitidos para os índices?
  - Range check? (verificação de índices)
  - Local da alocação?
  - Forma retangular ou irregular?
  - Etc

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Índices: tipo e verificação de faixa

- C, Java e Fortran: apenas inteiros
- Pascal, Ada: qualquer tipo ordinal (e.g. enumeração)

- C, C++, Perl e Fortran não fazem verificação de faixa.

```
int v[10];
v[11] = 0;
```

- Java, ML e C# fazem
  - execução aumenta a confiabilidade, mas perde eficiência

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Alocação / Vinculação de armazenamento

Categoria de Vetor	Tamanho	Tempo de Definição	Alocação	Local de Alocação
Estáticos	Fixo	Compilação	Estática	Base
Semi-Estáticos	Fixo	Compilação	Dinâmica	Pilha
Semi-Dinâmicos	Fixo	Execução	Dinâmica	Pilha
Dinâmicos	Variável	Execução	Dinâmica	Heap

```
// Em C:
void f() {
   static int v[10]; }
```

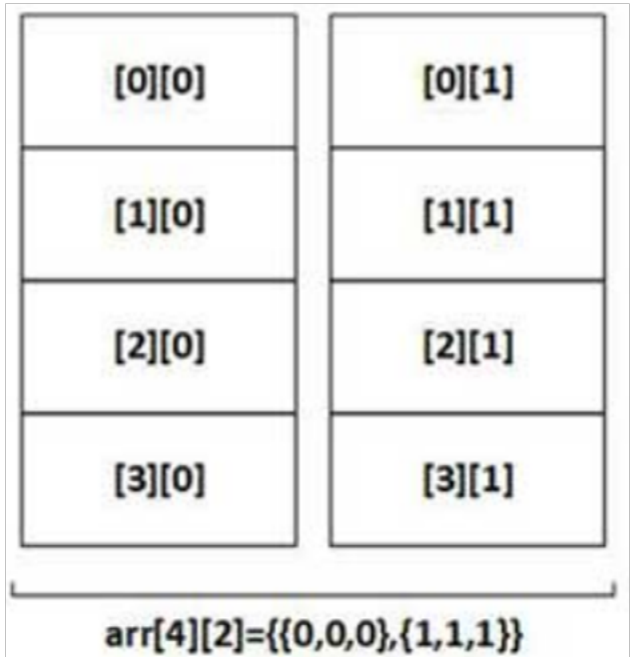
```
// Em C:
void f() {
   int v[10]; }
```

```
// Em C ISO/99:
void f(int n) {
   int v[n]; }
```

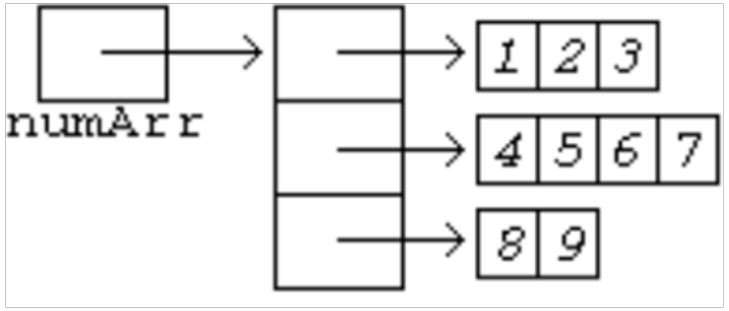
```
// Em C++:
void f(int n) {
   int v[] = new int [n]; }
```

Tipos de Array

Rectangular array



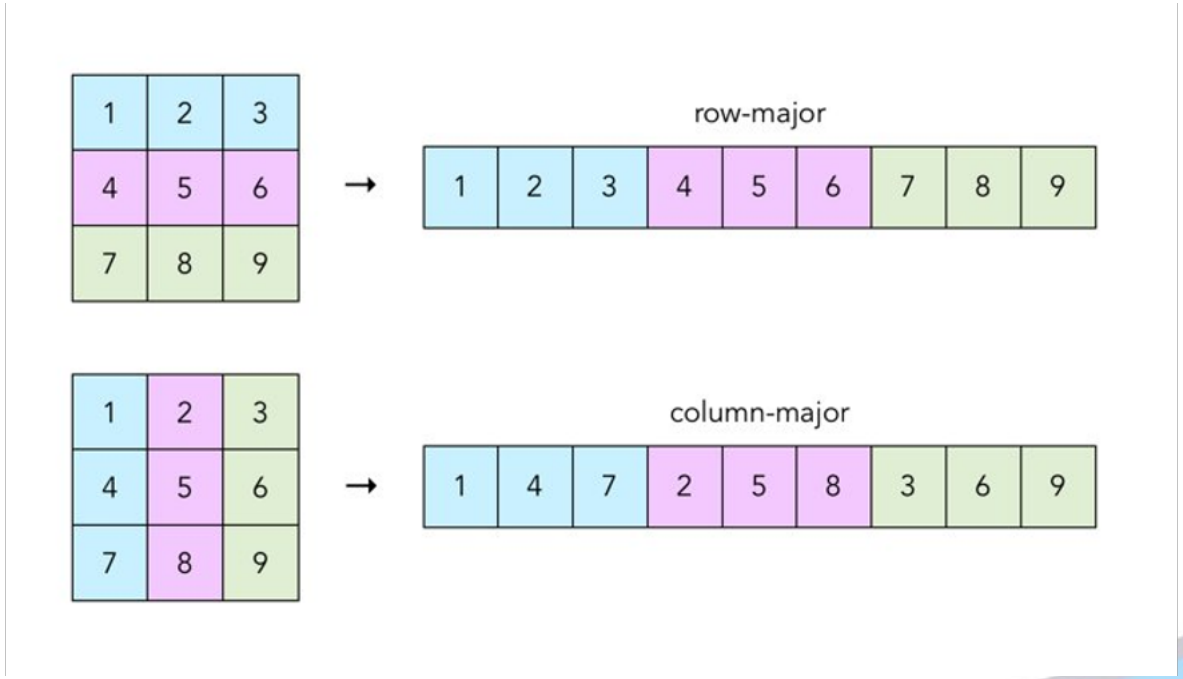
Jagged array




```
int[][]c;
c = new int[2][]; // creates 2 rows
c[0] = new int[5]; // 5 columns for row 0
c[1] = new int[3]; // create 3 columns for row 1
```

Layout de memória em arrays

- *Row major order* (ordem principal de linha)
- *Column major order* (ordem principal de coluna)



Layout de memória em arrays

- Muito importante!
- Impacto muito considerável no desempenho
- Matriz abaixo tem tamanho 10.000 x 10.000 (código C)

```
for (int j = 0; j < SIZE; j++) {
    for (int i = 0; i < SIZE; i++) {
        a[i][j] = 42;
    }
}
```

Time: 1.4 s

```
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        a[i][j] = 42;
    }
}
```

Time: 0.146 s

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Questionável!

- "Column major order (ordem principal de coluna) não é empregada em nenhuma linguagem comumente utilizada.” -Sebesta (11a. edição)

- Column-major order é utilizada em Fortran, MATLAB, GNU Octave, R, Julia, etc.

- https://en.wikipedia.org/wiki/Row-_and_column-major_order

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Matrizes Associativas

- Uma matriz associativa é um conjunto não ordenado de elementos indexados por um número igual de valores chamados chaves

- Perl:
- %hi_temps = ("Mon"=>77, "Tue"=>79, “Wed”=>65, …);

- Python (Dicionários)
- hi_temps = {“Mon”:77, “Tue":79, “Wed”:65}

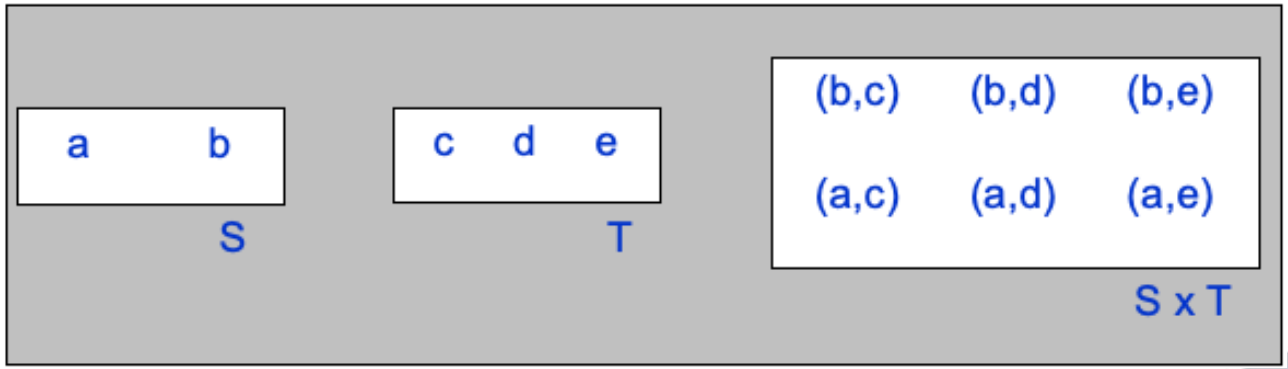
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos Compostos

- Criados a partir de tipos simples/primitivos
- Entendidos em termos dos seguintes conceitos (Watt 1990):
  - Produto cartesiano; struct C
  - Uniões; union C
  - Mapeamentos; arrays, dicionários ou funções
  - Conjuntos potência; poucas linguagens
  - Tipos recursivos. nó aponta pra outro nó

Produto cartesiano

- Combinação de valores de tipos diferentes em tuplas:
- Exemplos:
  - registros de Pascal, Modula-2, Ada e COBOL
  - estruturas de C

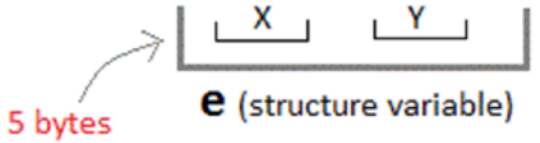


S x T x U ...
S x S x T x U

Registros (Records)

- Permite armazenar qualquer combinação de valores dos tipos especificados na ordem indicada:

```
struct Emp
{
 char X ;     // size 1 byte
 float Y ;    // size 4 byte
} e ;
```



- Existe em praticamente todas LPs
- Exemplo em C:

```
4  typedef struct {
5      char a;
6      int b;
7      float c;
8  } tipoStruct;
9
10 tipoStruct tipo1;
11
12 tipo1.b = 77;
```

tipoStruct = char x int x float

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tuplas (Tuples)

- Tuplas ⇒ Registros sem nome para os campos

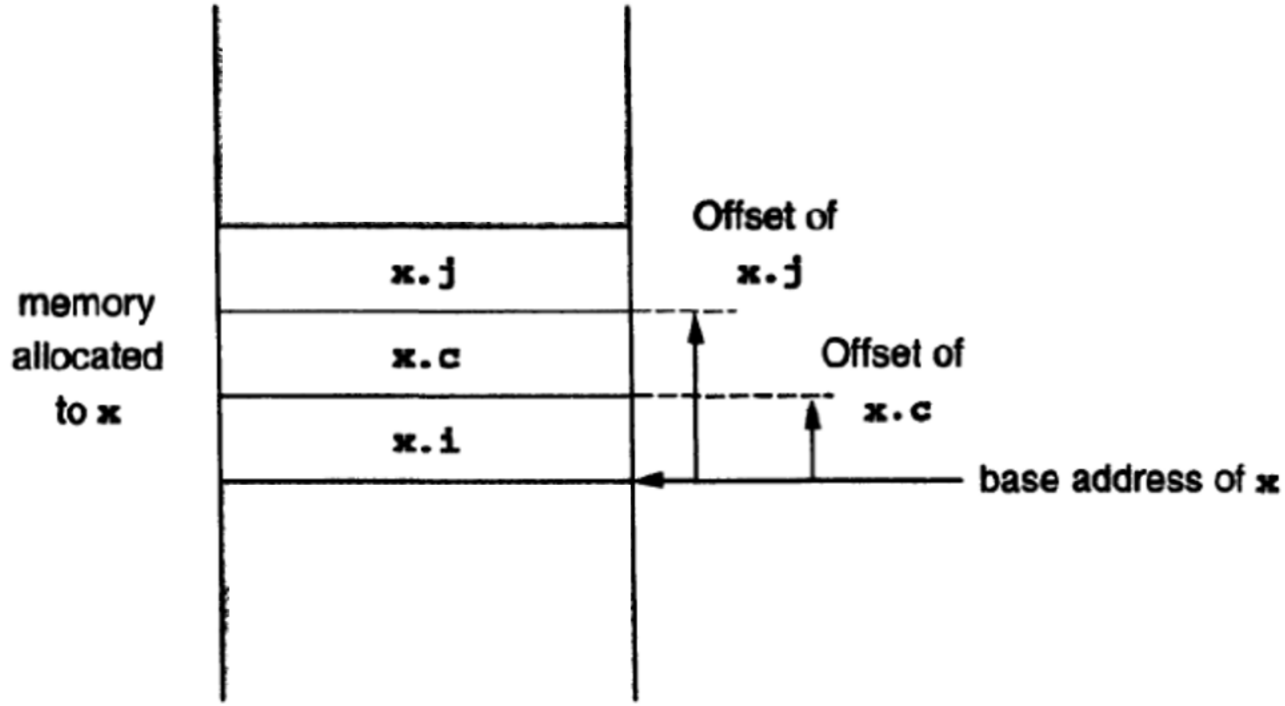
```
('a', 42, 4.2) = char x int x float
```

- Ou registros são tuplas com nomes para acessar seus elementos :)

Acesso em registros

- Compilador calcula offsets dos campos

```
typedef struct rec
   { int i;
     char c;
     int j;
   } Rec;
...
Rec x;
```



Acesso em arrays x registros

- Array: Quase sempre exigem operações em tempo de execução

$$\texttt{list[k]}$$

$$\text{address}(\texttt{list[k]}) = \text{address}(\texttt{list[0]}) + \texttt{k} * \text{element\_size}$$

- Registros: Quase sempre podem ser determinados em tempo de compilação

```
employee.name = "Freddie"
employee.hourlyRate = 13.20
```

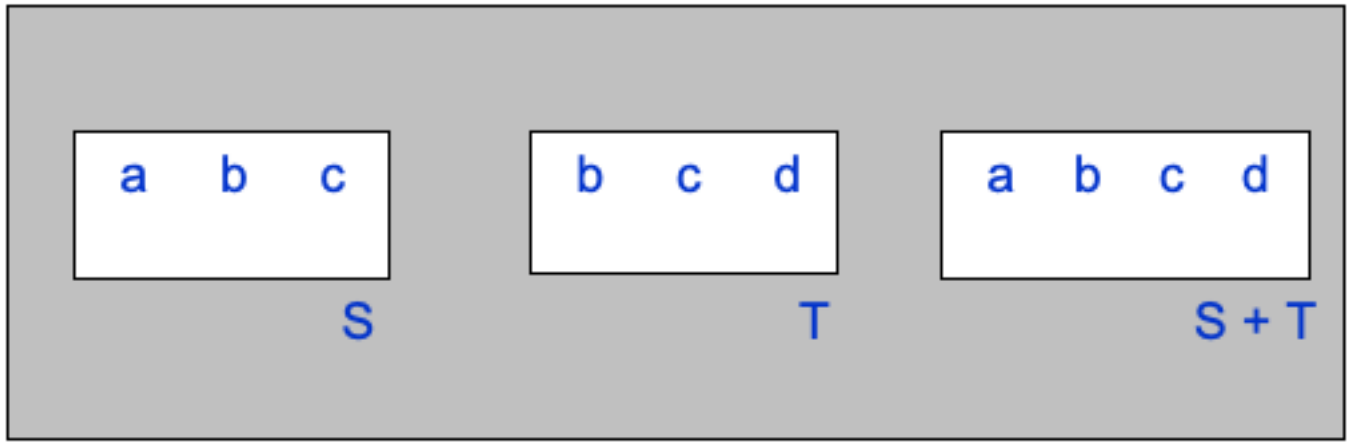
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos Compostos

- Criados a partir de tipos simples/primitivos
- Entendidos em termos dos seguintes conceitos (Watt 1990):
  - Produto cartesiano; struct C
  - Uniões; union C
  - Mapeamentos; arrays, dicionários ou funções
  - Conjuntos potência; poucas linguagens
  - Tipos recursivos. nó aponta pra outro nó

Uniões (Unions)

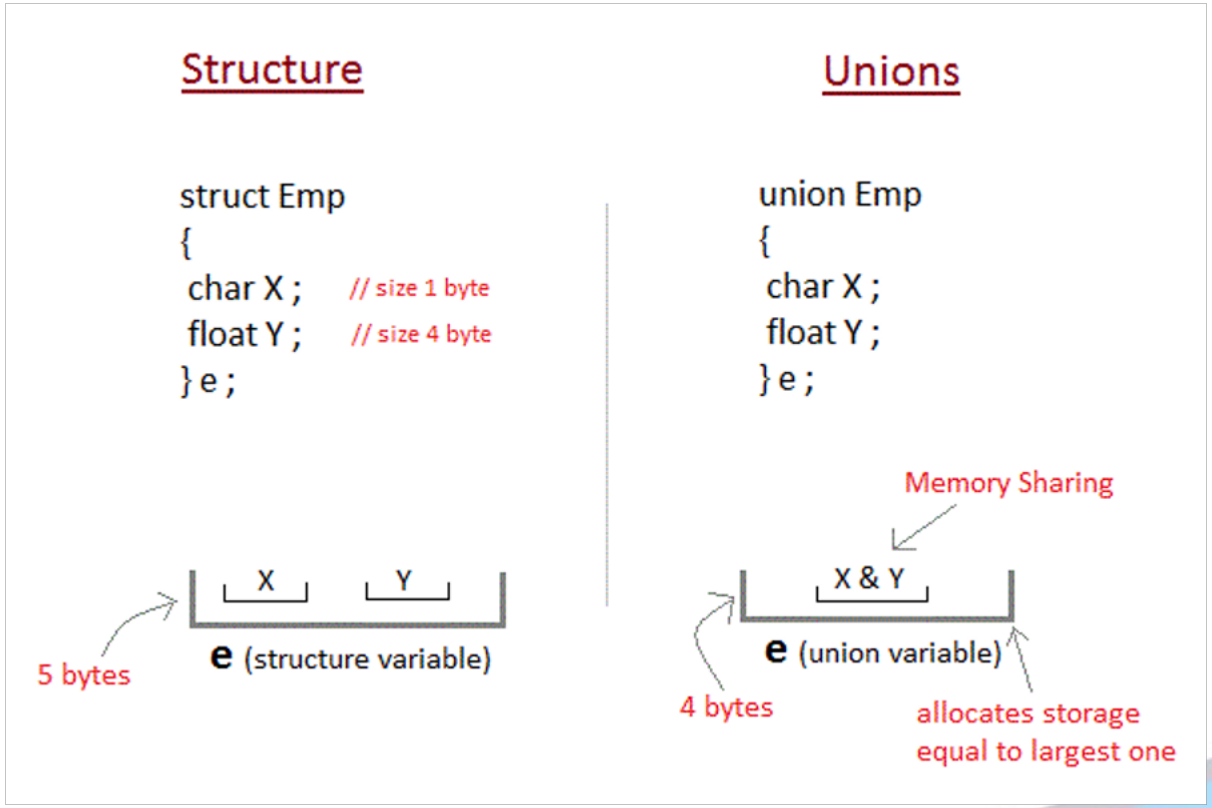
- Consiste na união de valores de tipos distintos para formar um novo tipo de dados:



S + T + U ...

Uniões (Unions)

- Estrutura com tipo variável em tempo de execução



Uniões (Unions)

- Uniões livres:
  - Interpretação “livre” do dado armazenado
  - Exemplos:
    - union de C
    - equivalence de Fortran
  - Permite subverter o sistema de tipos
  - Java decidiu não suportar

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

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

- Uniões disjuntas/discriminadas:
  - possuem tag identificadora de tipo
  - Interpretação restrita do dado armazenado
  - Exemplos:
    - registros variantes de Pascal,
    - union de Ada e ALGOL 68.

```
(* Exemplo em Pascal: *)
TYPE   Representacao = (decimal, fracionaria);
  Numero = RECORD
  CASE Tag: Representacao OF
        decimal: (val: REAL);
        fracionaria: (numerador, denominador:
      INTEGER);
  END;
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

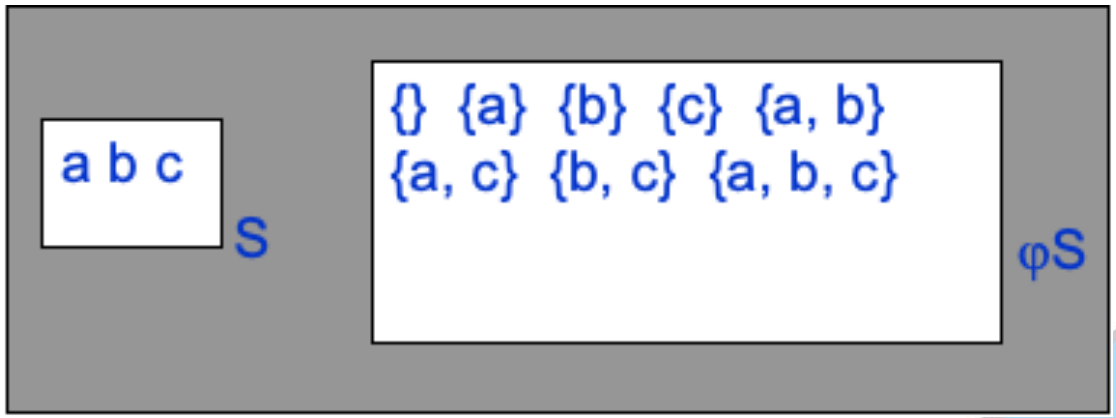
Tipos Compostos

- Criados a partir de tipos simples/primitivos
- Entendidos em termos dos seguintes conceitos (Watt 1990):
  - Produto cartesiano; struct C
  - Uniões; union C
  - Mapeamentos; arrays, dicionários ou funções
  - Conjuntos potência; poucas linguagens
  - Tipos recursivos. nó aponta pra outro nó

Conjuntos potência

- Poucas LPs oferecem. Muitas vezes de forma restrita;
- Pascal (somente discretos, primitivos e pequenos):

- Operações básicas:
  - Pertinência;
  - Contém;
  - Está contido;
  - União;
  - Diferença;
  - Diferença simétrica;
  - Interseção.



var Cores : set of {verde, vermelho, amarelo}

enum {verde, vermelho, amarelo} cor ???

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos recursivos

- 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;
}
```

Ponteiros e Referências

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Ponteiros e Referências

- Tipos cujos valores correspondem a endereços de memória.

- **Ponteiros** podem conter um endereço de memória, null ou um valor qualquer (lixo). Seus valores podem ser manipulados livremente, inclusive com operações aritmética como no caso de arrays em C.

- **Referências** são como *apelidos(alias)* para objetos/valores existentes. Implicitamente são também ponteiros mas com restrições nas operações: só desreferenciação (acesso ao valor/ objeto apontado)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Ponteiros e Referências

- Problemas típicos do uso de ponteiros:
  - Ponteiro solto (dangling pointers)
    - Um ponteiro que contém o endereço de uma variável dinâmica no monte desalocada
  - Variável dinâmica do monte perdida
    - Não há (mais) ponteiro que aponte para tal endereço
    - Vazamento de memória (*memory leakage*)

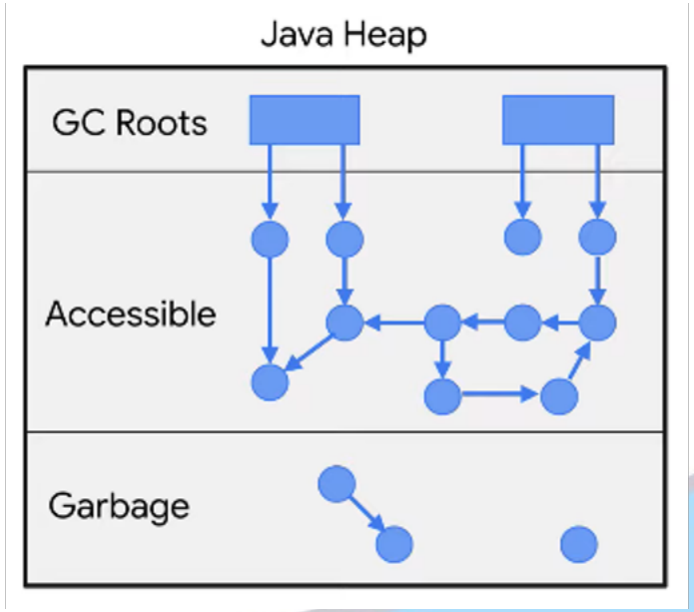
Ponteiros e Referências

- Livro menciona ponteiros e referências pendentes/ soltas/dangling
- Podem acontecer com a desalocação de uma variável no heap (exemplo da seção 6.11.3.1)
- Mas também pode acontecer na pilha!

```
int* dangle(void) {
  int x;
  return &x;
}
```

Lixo de memória (garbage)

- Coleta de lixo (garbage collection)
  - Desalocação implícita de áreas do heap que não são mais acessíveis
- Exemplo: LISP e Java
- Abordagens:
  - Marcar e varrer
  - Contador



Vazamento de memória (memory leakage)

- Memória inacessível por erros no programa

```
void leak()
{
    int **list = new int*[10];
    for (int i = 0; i < 10; i++)
    {
        list[i] = new int;
    }

    delete list;
    return;
}
```

Leak !

- Allocation a series, delete only one unit

Solution

```
for (int i = 0; i < 10; i++)
{
    delete list[i];
}
delete list;
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Falácias de memory leakage

- Não precisa desalocar em C/C++, deixa que o SO cuida quando o processo terminar...
  - E se o seu programa precisar rodar por dias?

- Não existe memory leakage em linguagens com coletor de lixo...
  - Seu programa pode atrapalhar o coletor

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Falácias de memory leakage

- Para saber mais:
  - Java vs C++: Trading UB for Semantic Memory Leaks (Same Problem, Different Punishment for Failure)
  - http://ithare.com/java-vs-c-trading-ub-for-semantic-memory-leaks-same-problem-different-punishment-for-failure/

Verificação de Tipos, Equivalência e Coerção

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Verificação de Tipos (*Type checking*)

- Garante que todas as expressões possuem tipos compatíveis
- Se não forem ⇒ erro de tipo (type error)

- Atribuição ou comparação, essa expressão só é valida se os tipos de x e y são compatíveis:

```
x = y
```

  - São equivalentes, ou
  - É possível coerção entre os tipos (primitivos)

- Verificação de tipos pode ser
  - Estática (C, C++, etc)
  - Dinâmica (Python, JS, etc)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Equivalência de Tipos

- Critérios diferentes para equivalência de tipos:
  - Sistema de tipos estrutural:
    - Tipos são equivalentes se tem a mesma estrutura.
  - Sistema de tipos nominativo
    - Tipos são equivalentes se tem mesmo nome
    - Em C, os registros seguintes não são compatíveis:

```
struct Tree {
  struct Tree *left;
  struct Tree *right;
  int value;
};
```

```
struct STree {
  struct STree *l;
  struct STree *r;
  int number;
};
```

Coerção (*Coercion*)

- Coerção implícita: feita automaticamente pelo compilador ou interpretador em expressões mistas:

```
int i = 1;
float f = 1.0;
if ((i + f) == 2)
   printf("Type coercion worked!");
```

- Coerção explícita: feita pelo programador:

```
int i = 1;
float f = 1.0;
if ((i + (int)f) == 2)
   printf("Type coercion not necessary!");
```

Coerção (*Coercion*)

- Sistema de tipos da LP define conversões válidas
- Exemplo de adição em Fortran 77

+	integer	real	double	complex
integer	integer	real	double	complex
real	real	real	double	complex
double	double	double	double	*illegal*
complex	complex	complex	*illegal*	complex