

Linguagens de Programação

Tipos de Dados

Baseado em Conceitos de Linguagens de Programação – Robert W. Sebesta

Prof. Lucas Ismaily

Universidade Federal do Ceará
Campus Quixadá

Evolução

- Evolução de Tipos de Dados:
 - FORTRAN I (1956) - INTEGER, REAL, arrays
 - ...
 - Ada (1983) - Utilizadores podem criar novos tipos de dados.
- Um descritor é um conjunto de atributos de uma variável (Nome, Endereço, Valor, Tipo, Tempo de vida, Escopo).
- Questões de Projecto para tipos de dados:
 - Como especificar variáveis?
 - Quais as operações que estão definidas sobre determinada variável?

Tipos de Dados Primitivos

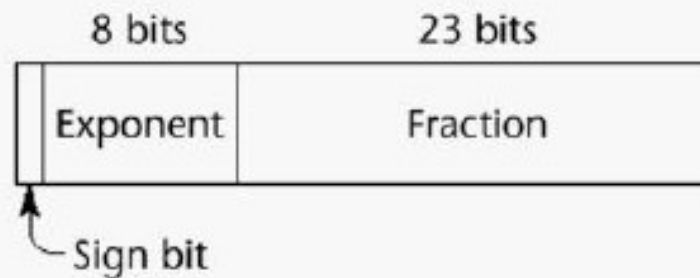
- Tipos Primitivos de Dados são todos os tipos que não são definidos em termos de outros tipos.
- Inteiro (*integer*)
 - Em geral reflete a situação do *hardware* (registros)
 - Existem vários tipos de números inteiros numa linguagem:
inteiro com/sem sinal, inteiro base decimal, inteiro base binária, precisão simples, etc.

Tipos Primitivos - Ponto Flutuante

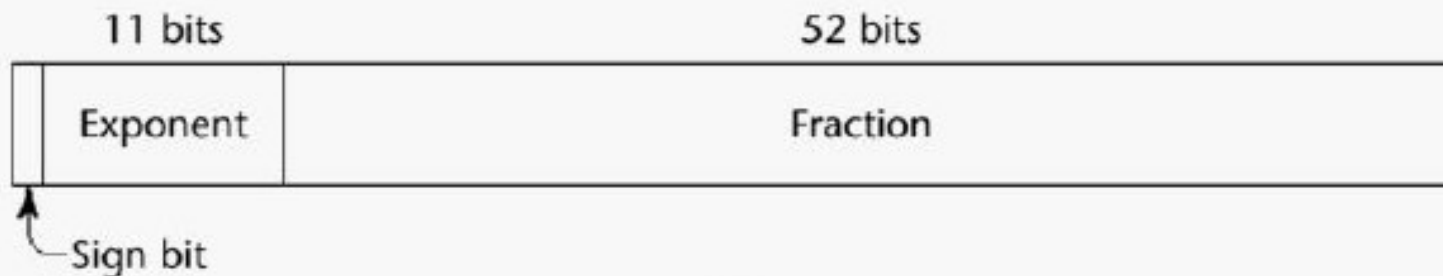
- Ponto Flutuante (*Floating Point*)
 - Aproximação ao conjunto dos números reais
 - Linguagens científicas suportam pelo menos dois tipos de pf: precisão simples e estendida
 - Relação próxima com o hardware. Algumas linguagens permitem expressar precisão, como por exemplo em Ada:
 - type SPEED is digits 7 range 0.0..1000.0;
 - type VOLTAGE is delta 0.1 range -12.0..24.0;

Tipos Primitivos - Ponto Flutuante

- Formatos de Representação de Pontos Flutuantes:
(a) Precisão Simples, (b) Precisão Estendida



(a)



(b)

Tipos Primitivos - Decimal

- Decimal (*Base 10*)
 - Para aplicações comerciais
 - Armazena um número determinado de dígitos decimais (codificado na base binária)
 - Vantagem:
 - Precisão.
 - Desvantagens:
 - Amplitude limitada (*range*), grande gasto de memória.

Tipos Primitivos - Lógico

- Lógico (*Boolean*)
 - Só permite dois valores: Verdadeiro ou Falso (*true or false*);
 - Pode ser implementado num bit, mas geralmente é implementado num byte.
 - Vantagem:
 - Legibilidade.

Tipo String de caracteres

- String
 - Valores são sequências de caracteres
 - Considerações de Projecto:
 - Tipo primitivo ou tipo especial de sequência?
 - Dimensão é estática ou dinâmica?
 - Operações:
 - Atribuição;
 - Comparação (=, >, <, etc.);
 - Concatenação (junção ao fim da string);
 - Referência a substring;
 - Correspondência de padrão (*Pattern matching*).

Tipo String de caracteres

- Exemplo do Pascal:
 - tipo não primitivo;
 - operação de atribuição e comparação somente (em packed arrays).
- Exemplo do Ada, FORTRAN 77, FORTRAN 90 and BASIC
 - Mais ou menos primitivo;
 - Atribuição, comparação, concatenação, referência a substring;
 - FORTRAN possui correspondência de padrão intrínseco.

Tipo String de caracteres

- Exemplo em Ada:
 - N := N1 & N2 (concatenação)
 - N(2..4) (referência a substring)
- Exemplo do C, C++:
 - Não primitivo;
 - Usa char arrays e biblioteca de funções para as operações.
- Exemplo do SNOBOL4 (Ling. para tratamento de string)
 - Primitivo;
 - Muitas operações, incluindo um elaborado método para correspondência de padrão (*Pattern matching*).

Tipo String de caracteres

- Exemplo em Perl:
 - Padrões são definidos em termos de expressões regulares;
 - Grande poder de expressão!;
 - Ex. de expressão regular:
`/[A-Za-z]+/` → Palavras contendo unicamente letras.
- Exemplo do Java:
 - String class (não é arrays de char).

Tipo String de caracteres

- Opções na definição do tamanho da *String*:
 - Estático - FORTRAN 77, Ada, COBOL
Ex. FORTRAN 90:

```
CHARACTER (LEN = 15) NAME;
```
 - Dinâmico Limitado – C e C++:
 - Tamanho real indicado por um carácter nulo
Ex. C++:

```
char * buffer;
```

```
buffer = new char[512];
```
 - Dinâmico – SNOBOL4, Perl

Tipo String de caracteres

- Avaliação:
 - Ajuda a escritabilidade do programa
 - O tipo primitivo de tamanho estático é eficiente. Porque não o ter?
 - Tamanho dinâmico é bom, mas muito caro. Será que não vale a pena implementá-lo?

Tipo String de caracteres

- Implementação:
 - Tamanho estático - Descritor é definido e utilizado em tempo de compilação.
 - Tamanho dinâmico limitado - pode necessitar de um descritor em tempo de execução para o tamanho (não acontece em C e C++).
 - Dinâmico - precisa de um descritor em tempo de execução; Reserva/liberação de memória é o grande problema da implementação.

Tipo Ordinal

- Tipo Ordinal
 - Um tipo ordinal é um tipo em que a amplitude dos possíveis valores podem facilmente ser associados com os inteiros positivos;
 - Em muitas linguagens os programadores podem definir os seguinte tipos ordinais:
 - Tipo Enumerado;
 - Tipo Subfaixa.

Tipo Ordinal - Enumerados

- Tipos enumerados - tipo onde o programador enumera todos os possíveis valores através de constantes simbólicas.
 - Exemplo em Pascal:
`type cor = (vermelho, azul, branco, preto);`
 - Exemplo em C e C++:
`enum cor {vermelho, azul, branco, preto};`
 - Considerações de Projecto:
 - Pode uma constante simbólica pertencer a mais de uma definição de tipo?

Tipo Ordinal - Enumerados

- Exemplos do Pascal:
 - Constantes não podem ser reutilizadas.
 - Podem ser usadas em subscritos de arrays, como variável de ciclo *for* e selecção *case*, podem ser comparadas mas não podem ser usadas em E/S.
 - Ex.:

```
type tipocor = (Vermelho, Azul, Verde, Amarelo);  
Var  
  cor : tipocor;  
  ...  
  cor := Azul;  
if (cor > Vermelho)then ...
```

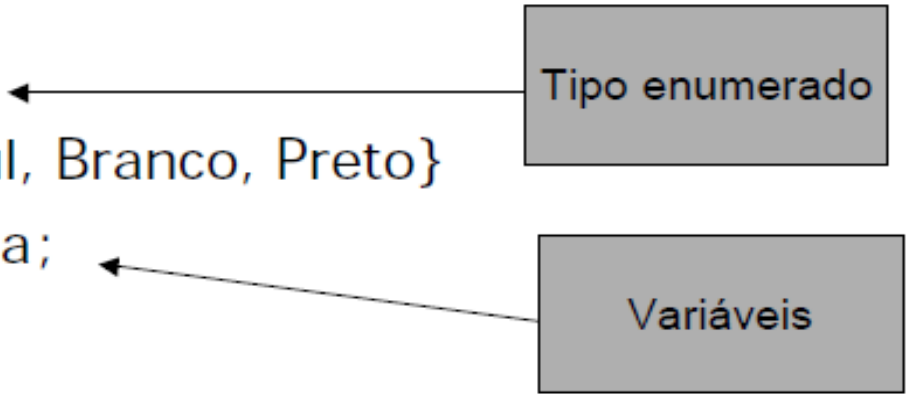
Tipo Ordinal - Enumerados

- Exemplo em C and C++ :
 - Como em Pascal, excepto que podem participar de E/S como inteiro.

■ Ex.:

```
enum tipocor  
{Vermelho, Azul, Branco, Preto}  
casaco, camisola;  
casaco = Azul;
```

Tipo enumerado



Variáveis

- Exemplo do Java
 - Não possui tipo enumerado

Tipo Ordinal - Enumerados

- Avaliação:
 - Melhora a legibilidade, uma vez que, não é necessário modelar cor como um número inteiro;
 - Melhora a confiabilidade, dado que o compilador pode verificar a validade das operações e amplitude dos valores em tempo de compilação.

Tipo Ordinal - Subfaixa

- Tipo Subfaixa (Subrange) - Subsequência ordenada de um tipo enumerado ordinal.
 - O tipo subfaixa foi introduzido pelo Pascal, e posteriormente, foram incluídos também em Ada e Modula-2.
 - Ex.: os números 12..14 são uma subfaixa dos n^ºs inteiros.
- Decisão de Projecto:
 - Como os valores podem ser usados?

Tipo Ordinal - Subfaixa

- Exemplo em Pascal :
 - Subfaixa comporta-se como o seu tipo pai. Podem ser usados em arrays, comparação, ciclos, etc..
 - ex. `type positivo = 0 .. MAXINT;`
 `type maiusculas = 'A'..'Z'`
- Exemplo em Ada:
 - Subfaixa é um subconjunto do tipo pai (não é um tipo novo), logo são compatíveis. Podem ser usados como em Pascal. Ex.:
`subtype positivo is INTEGER range 0..INTEGER'LAST;`

Tipo Ordinal

- Avaliação de tipos enumerados:
 - Melhor legibilidade;
 - Melhor confiabilidade - detecção de erros e das amplitudes dos valores.
- Implementação:
 - Tipos enumerados geralmente são implementados associando números inteiros a cada constante.
 - Subfaixas são implementados da mesma forma que os tipos pais.
 - O código para restringir atribuições a variáveis de subfaixas deve ser inserido pelo compilador.

Tipo Array

- **Array** - é uma sequência de dados homogênea, em que cada elemento é uma variável e pode ser acessado por sua posição relativa, a partir da origem (primeiro elemento).
- É uma estrutura de dados fundamental e existente em todas as linguagens.

Tipo Array

- Considerações de Projecto:
 - Que tipo é correcto para índices?
 - Haverá verificação de índices?
 - Quando as amplitudes dos índices são vinculadas?
 - Quando reservar espaço?
 - Qual o número máximo de elementos?
 - Sequências podem ser inicializadas?
 - Arrays podem ser divididos em subfaixas?

Tipo Array

■ Indexação

- Associação entre índices e elementos de um array.
- `map(nome-array, valor-índice) → elemento`
- Sintaxe para os índices:
 - FORTRAN, PL/I, Ada - parênteses curvos. Ex.: `A(i)`
 - Em Fortran os parênteses são ambíguos:
`SOMA = SOMA + A(i) --> A(i)` pode ser a invocação de uma função ou acesso a um elemento de um array.
 - Maioria das outras linguagens utilizam parênteses rectos. ex.: `A[i]`

Tipo Array

- Tipo de índices:
 - FORTRAN, C - somente números inteiros;
 - Pascal - qualquer tipo ordinal escalar:
int, boolean, char, enum;
 - Ada - int, enum, boolean e char;
 - Java - unicamente números inteiros.
- Limite inferior dos índices:
 - C/C++ e Java = 0;
 - Fortran, Basic = 1;
 - Outras linguagens = deve ser especificado.

Tipo Array

- Categorias de Arrays – os arrays podem ser classificados segundo o tipo de vinculação de índices (subscritos) e armazenamento:
 - I. Array Estático;
 - II. Array Pilha-dinâmico de Tamanho Fixo;
 - III. Array Pilha-dinâmico;
 - IV. Array Heap-dinâmico.

Nota: Nomes indicam **onde** e **quando** são alojados.

Tipo Array – Categorias de Arrays

I. Array Estático

- Armazenamento do array é estático;
- Tamanho do array é estático;
- Índices e armazenamento são vinculados em tempo de compilação;
- Tipo de índice vinculado em tempo de projecto da linguagem (ex. Fortran = INTEGER).
- Vantagem: eficiência de execução (não há reserva nem liberação de memória).

Tipo Array – Categorias de Arrays

- Ex. em FORTRAN 77:

```
INTEGER INTLISTA(99)    // índices (1 .. 99)
```

- Armazenamento é alojado estaticamente.

- Ex. em C:

```
int bufferGlobal[50];    // índices (0 .. 50)
```

```
void funcaoX()
```

```
{ static int buffer[99];    // índices (0 .. 99)
```

```
    //...
```

```
}
```

Tempo de vida dos 2 buffer = tempo de vida da aplicação.

Tipo Array – Categorias de Arrays

II. Pilha dinâmica de Tamanho fixo

- Índices com vinculação estática;
- Armazenamento é vinculado em tempo de execução na pilha dinâmica;
- Dimensão é estática;
- Tempo de vida do array = tempo do bloco ou função;
- Vantagem: eficiência de espaço de memória

Tipo Array – Categorias de Arrays

III. Dinâmico de pilha

- Dimensão e armazenamento são dinâmicos porém assim que forem definidos são fixos durante o tempo de vida do array.
- Ex. em Ada: (declare blocks)

```
declare STUFF : array (1..N) of FLOAT;  
begin      ...  
end;
```
- Vantagem: flexibilidade - tamanho não precisa ser conhecido até que o array seja utilizado.

Tipo Array – Categorias de Arrays

IV. Dinâmico de Heap

- Dimensão e armazenamento são dinâmicos (e podem mudar qualquer número de vezes durante o seu tempo de vida).
- Tempo de vida controlado explicitamente pelo programador, ou implicitamente pelo *Garbage Collection*.
- Vantagem: flexibilidade – O tamanho pode crescer ou diminuir conforme a necessidade.

Tipo Array – Categorias de Arrays

- Ex. de Array Heap-dinâmico em C:

- Arrays dinâmicos são apontadores;
- Não existe verificação de dimensão;
- Funções padrão: *malloc*, *free*

```
void main( )
```

```
{ char *string; /* Define pointer for name */  
  string = malloc(1024); /* Allocate space */  
  free( string ); /* free memory space */  
}
```

Tipo Array

- Número de índices:
 - FORTRAN I => 1 a 3 dimensões;
 - Ex.: `buffer(5)(3)(1)`
 - FORTRAN 77 => até 7 dimensões;
 - C, C++, Java => somente uma dimensão, porem elementos podem ser arrays;
 - Ex.: `int matriz[5][4];` // array de arrays
 - Outras Linguagens => sem limite.

Tipo Array – Inicialização de Arrays

- Inicialização de arrays
 - Em geral os arrays são inicializados através de uma lista de valores, respeitando a ordem em que os elementos são armazenados
 - Ex. em Fortran:
 - Utilização do comando DATA e colocação dos valores entre / ... / na declaração
INTEGER LISTA(3)
DATA LISTA /0, 5, 5/
 - Ex. em Pascal e Modula-2
 - Não permitem inicialização de arrays

Tipo Array – Inicialização de Arrays

Ex. em C, C++:

- Valores serão dados entre { } separados por virgulas, e o compilador calculará a dimensão para o array.

```
int stuff[] = {2, 4, 6, 8};
```

```
/* array de 4 números inteiros */
```

```
char nome[] = "José";
```

```
/* array 4 caracteres finalizado com elemento nulo (zero) */
```

```
char *nomes[] = {"jose", "pedro", "Maria"};
```

```
/* array de string */
```

Tipo Array – Operações Arrays

- Operações com Arrays
 - Uma operação de array é uma operação que opera na estrutura como uma unidade.
 - Ex. em APL:
 - APL contém muitas operações definidas tanto para vetores como para matrizes.
 $A + B$, $A * B$, $A - B$
($*$, $+$, $-$, etc)) – operações definidas para qualquer tipo (escalar, vector e matrizes).

Tipo Array – Operações Arrays

- Ex. em APL (cont.):

outras operações:

- inverter elementos de um vector
- inverter colunas de uma matriz
- inverter linhas de uma matriz
- transpor uma matriz
- inverter uma matriz
- etc.

Tipo Array – Operações Arrays

- Fatias (*slices*)
 - Uma fatia é uma referência a uma parte de um array de qualquer dimensão.
 - Em geral as linguagens não suportam fatias de um array.
 - PL/I foi a primeira linguagem a introduzir este conceito em 1965.

Tipo Array – Implementação

- Implementação de Arrays
 - Transformação de índices (expressão) para o endereço de um elemento;
 - Armazenamento dos elementos do array por **linha** (ordem lexicográfica) ou por **coluna** (ordem anti-lexicográfica);

Tipo Array Associativo

- Array Associativo (*Associative Array*) - é uma colecção não ordenada de elementos indexados por chaves.
 - Problemas de projecto:
 - Qual o esforço para referenciar um elemento?
 - O tamanho do array é estático ou dinâmico?
 - O índice precisa ser armazenado.
 - OBS: Em arrays normais os índices são calculados, e por isso, não são armazenados.

Tipo Registro

- Um registo é um conjunto, possivelmente heterogéneo de elementos, onde cada elemento é designado por campo e identificado por um nome e seus atributos.
- Considerações de projecto:
 - Qual a forma de referenciar cada campo?
 - qualificação por ponto;
 - uso de parênteses;
 - total ou parcialmente qualificado
 - Que operações são definidas para o registo?

Tipo Registro

- Ex. de registro em COBOL:
 - Sintaxe: COBOL, PL/I utilizam números de níveis para aninhamento de registros.

01 Registro-empregado

02 Nome-Empregado

05 Primeiro Picture is X(20).

05 Meio Picture is X(10).

05 Ultimo Picture is x(20).

02 Salario Picture is 99v99.

Tipo Registro

- Ex. de registro em Pascal, Modula-2:
 - Sintaxe: Pascal, Modula-2 utilizam definições recursivas com aninhamento de declarações.

```
Reg_Emp : record
  Nome: record
    Apelido : string(1..10);
    nproprio : string(1..10);
  end;
  Salario : real;
end;
```

Tipo Registro

- Ex. de registro em C, C++:
 - Sintaxe: C, C++ não permite aninhamento.

```
struct Reg_Emp  
{  
    char Primeiro(20);  
    char Meio(10);  
    char Ultimo(20);  
    double Salario;  
};
```

Tipo Registro

- Referências a campos do registro –
sintaxe para especificar e aceder a cada
um dos campos de um registo.
 - Sintaxe do COBOL
field_name OF record_name_1 OF ... OF
record_name_n
 - Sintaxe das outras L.P. (notação ponto)
record_name_1.record_name_2. ...
record_name_n.field_name

Tipo Registro

- Tipos de referências:
 - Referências totalmente qualificadas devem incluir todos os nomes de registros aninhados, finalizando com o nome do campo.
 - Referências parcialmente qualificadas (*elíptica*) permitem descrever apenas alguns nomes de registros, desde que não exista ambiguidade de referência.
 - Pascal e Modula-2 possuem cláusula *with* para simplificar referências

Tipo Registro

- Ex. registo e cláusula with em Pascal:

```
type reg_empregado =  
    record  
        nome : string (40);  
        sexo: (Masculino, Feminino);  
    end;  
  
var  
    empregado : reg_empregado  
with empregado do  
    begin  
        nome := 'José';  
        sexo := Masculino;  
    end;
```


Tipo Registro – Operações

- Operações com o tipo Registro
 - Atribuição
 - Pascal, Ada, C: Se os tipos forem idênticos;
 - Em Ada, o lado direito pode ser um *constante agregada* (tipo especial de constante);
 - Cobol: instrução *move corresponding* para atribuição a campos de mesmo nome.
 - Inicialização
 - Permitido em Ada, usando constantes agregadas.

Tipo Registro – Operações

- Comparação
 - Ada: '=' e '/=' (diferente); operando pode ser constante agregada
- MOVE CORRESPONDING
 - COBOL: Copia todos os campos de um registro fonte para os campos de mesmo nome de um registro destino.

Tipo Conjunto

- Um conjunto é um tipo de dados cujas variáveis podem armazenar uma colecção não ordenada de valores distintos de algum tipo ordinal, designado de tipo básico.
 - Considerações de projecto:
 - Qual o máximo número de elementos?
 - Quais as operações disponíveis sobre o conjunto?
 - Ex. de Java - possui uma classe para operações sobre conjuntos.

Tipo Conjunto

- Ex. em Pascal:
 - Sem tamanho máximo na definição da linguagem; Implementações geralmente impõe um limite.
 - Operações sobre conjuntos:
 - união '+', intersecção '*', diferença '-', comparação '=', superconjunto '>=', subconjunto '<=', pertinência 'in'.

```
type cores = ( azul, verde, amarelo, branco);  
        conjcores = set of cores;  
var c1, c2 : conjcores;  
if azul in c1 then ...  
if ch in ['a', 'e', 'i', 'o', 'u'] then ...
```

Tipo Conjunto

- Avaliação:
 - Aumenta a legibilidade;
 - Se uma linguagem não possui o tipo conjunto, este pode ser simulado.
 - Arrays são mais flexíveis que conjuntos, dado disponibilizarem muitas mais operações.

Tipo Apontador

- Considerações de projecto:
 - Qual o escopo e o tempo de vida de uma variável do tipo apontador?
 - Qual o tempo de vida de uma variável dinâmica de heap? (alojada/liberada pelo programador)
 - Apontadores são restritos a tipos específicos?
 - Podem ser utilizados para gestão dinâmica de memória?
 - Pode ser utilizado para endereçamento indirecto?
 - A linguagem deve suportar o tipo apontador, o tipo referência ou ambos?

Tipo Apontador

- O processo de perder variáveis heap-dinâmicas é chamado de "vazamento de memória" (*memory leakage*).
- Operações fundamentais de apontadores:
 - Atribuição de um endereço a um apontador;
 - Referência (diferenciação entre referência explícita e implícita).

Tipo Apontador

- Problemas com apontadores:
 - Referência oscilante (Dangling pointers):
 - Um apontador pode conter um endereço de uma variável heap-dinâmica que já foi liberada.
 - Geração de lixo (perca do endereço):
 - Uma variável heap-dinâmica quando não é mais referenciada por nenhum apontador, deixa de ser acessível.

Tipo Apontador

- Ex. de referência oscilante em C, C++:

```
int *apt1, *apt2;  
apt1 = (int *) malloc(sizeof( int ));  
apt2 = apt1;  
free(apt1);
```

Tipo Apontador

- Ex. de geração de lixo em C, C++:

```
char *apt1;
```

```
apt1 = (char *) malloc(sizeof( char ));
```

```
apt1 = (char *) malloc(sizeof( char ));
```

Tipo Apontador

- Ex. de C, C++:
 - Apontadores são utilizados principalmente para gestão dinâmica de memória e endereçamento;
 - Existência de operador para *desreferência* '*' e para *endereço-de* '&';
 - Pode-se efectuar aritmética de apontadores, em forma restrita;
 - Podem ser definidos apontadores genéricos 'void *' com ausência de tipo (não podem ser desreferenciados).

Tipo Apontador

- Ex. de definição de apontadores em C, C++:

```
int soma;
```

```
float stuff[100], * goods;
```

```
goods = stuff;
```

```
// *(goods+5) equivalente a stuff[5] e goods[5]
```

```
void * apt; // pode apontar qualquer tipo
```

```
apt = & soma; // ok - aponta para int
```

```
apt = goods; // ok - aponta para float
```

Tipo Apontador

- Tipo referência em C++ é um apontador que implicitamente denota o "endereços-de" variável/objecto.
 - Usado para passagem de parâmetros no modo de transmissão por referência (variável de entrada e/ou saída).
 - Referências vêm simplificar a utilização de apontadores.

Tipo Apontador

- Ex. de referências em C++:

```
int result = 0;
```

```
int &ref_result = result;
```

```
// neste caso: result e ref_result são aliases
```

```
ref_result = 100;
```

É equivalente a:

```
int result = 0;
```

```
int *ref_result ;
```

```
ref_result = &result;
```

```
*ref_result = 100;
```

Tipo Apontador

- Ex. de referências em Java:
 - Programador não manipula apontadores;
 - Não existe aritmética de apontador;
 - Não há liberação explícita de memória, colector de lixo ("*garbage collection*") é utilizado;
 - Não existe referências ambíguas;
 - Desreferenciação é sempre implícita;

Tipo Apontador

- Avaliação dos apontadores:
 - Referências perigosas e lixos são problemas assim como a gestão do heap;
 - Apontadores são como goto's para posições de memória - geram vários modos de acesso a uma célula de memória;
 - Apontadores são necessários - não é possível projectar uma linguagem sem eles.