|  |
| --- |
| **CAPITULO 05 – tipos de dados** |

Tipos de dados são os tipos surgiram há mais de meio século em linguagens de programação. Nas linguagens mais antigas era necessário implementar arvores binárias em arrays, apenas com o surgimento de ADA e COBOL pudemos manipular estruturas compostas/abstratos de dados.

1. **Tipos primitivos**

Tipos primitivos são os tipos que não são definidos em termos de outros tipos. Alguns tipos de dados podem ser reflexos do hardware, como por exemplo o **inteiro**.

* 1. **Tipos numéricos**

O tipo de dado numérico mais comum é o inteiro, que as vezes pode ser simples reflexo do hardware ou não tão simples, como em Pyton, onde temos o Long Int. Para representar um inteiro normalmente se usa uma string de bits em complemento de dois. Em C existe vários tipos de int, como o unsigned int, long int, long long int...

Já o tipo Ponto flutuante é o tipo real e apresenta problemas ao se representar o numero 0,1 por exemplo, pois é guardado em binário no computador. A maioria das linguagens apresenta o float como ponto flutuante de 4 bytes e o double como precisão dupla de 8 bytes.

Há ainda o tipo de dado booleano que na maioria das linguagens ocupa 1 byte. No C o tipo booleano não existe e expressões logicas com inteiro 0 são falsas e inteiro 1 são verdadeiras. Vantagem do tipo booleano é a legibilidade.

* 1. **Tipo caractere**

O código para caracteres mais usado é o ASCII, porém este sistema tem se tornado inadequado, novas linguagens como o Java, C# e Javascript tem usado o UNICODE para representar os dados.

* 1. **Tipo cadeia de caracteres**

Os tipos de dados strings de caracteres em C são representados em C por um vetor de char, que é finalizado por um caractere nulo , o que facilita que as strings tenham um tamanho dinâmico. Em Java as strings são suportadas pela classe String.

|  |
| --- |
| Questão de projeto: as strings devem ser um tipo de dado primitivo ou deve ser feito um array de char? |

Exemplos de implementação de strings:

|  |
| --- |
| 1. Uso de **strings estáticas** : uso de um descritor em tempo de compilação 2. Uso de **strings dinâmicas** : usa descritor em tempo de execução, o que resulta no problema de alocação. 3. Uso de **strings dinâmicas limitadas :**  podem precisar de um descritor em tempo de execução, porem no C/C++ isso não acontece devido o uso do caractere ‘/0’ |

OUTROS TIPOS : Complexo como Fortran e Pyton.

TIPO DECIMAL : importantíssimo para COBOL, vantagem: precisão, desvantagem: gasto de memória.

1. **Tipos enumeração**

São aqueles que todos os valores possíveis são enumerados na definição. Temos em enumerações uma melhoria na legibilidade do código.

OUTROS TIPOS: tipo faixa, melhora legibilidade e confiabilidade.

**3. Tipos matrizes**

Matrizes exigem índices que são custos a mais em tempo de execução. Em C/C++ são restritas a um mesmo tipo, em Ruby é apenas uma referência.

Decisões de projeto para as matrizes:

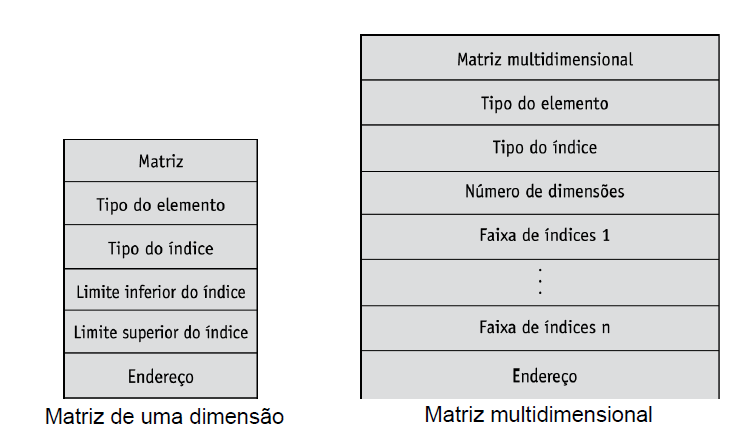
|  |
| --- |
| 1. Que tipos são permitidos para os índices? 2. Quando ocorre a liberação da matriz? 3. Matrizes multidimensionais irregulares são permitidas? |

A maioria das linguagens usa colchetes para indexar os elementos da matriz. Em C/C++ não é especificada a faixa de índices, por exemplo mat[-1] pode existir.temos as seguintes tipos de alocação de matrizes:

|  |  |  |
| --- | --- | --- |
| **static int** matriz[10][10] | **Vinculação estática** | Em C sem alocação dinâmica, eficiente |
| **int** matriz[10][10] | Dinamica de pilha fixa | Faixas vinculadas a pilha estáticamente |
|  | Dinamica da pilha | EX ada |
| Int\*\* matriz | Dinâmica de heap fixa | Em c, em vez da pilha usa-se o heap |
| ArrayList | Dinamica de heap | Pode mudar várias vezes durante a execução |

Matriz heterogênea é aquela que não precisa ser do mesmo tipo. Implementação de matrizes: se comparado aos dados primitivos exige um esforço adicional e Execução e compilação, exige também um gasto a mais para gerar o código que indexa a matriz.

Descritores para matrizes em tempo de compilação:



**MATRIZES ASSOCIATIVAS:** são tipos de matrizes onde chaves endereçam dados não ordenados, exemplo matriz do php.

1. **Tipos REGISTROS**

É um agregado de elementos individuais o qual os elementos individuais são acessados por nomes. É acessado por deslocamento a partir do inicio do registro.

Questões de projeto:

|  |
| --- |
| 1. Como acessar os campos? 2. Tem **ortogonalidade(registros de registros)?** |

Os registros são mais rápidos que as matrizes pois os campos deles são estáticos.

1. **UNIONS**

São um tipo em que as variáveis podem armazenar diferentes valores com diferentes tipos durante a execução do programa. Em C/C++ as unions são ditas livres pois não existe suporte para a verificação de tipos. As unions são potencialmente inseguras pois possivelmente não permitem verificação de tipos. Java e C# não tem unions, devido a preocupação com segurança.

1. **Ponteiros e Referencias**

**6.1. Referencias**

Um ponteiro é um tipo no qual as variáveis possuem uma faixa de valores que consistem em endereços de memória válidos e um valor especial NULL/NIL/null. Fornecem uma maneira de gerenciar a memória dinâmica, a heap.

PROBLEMA DOS PONTEIROS SOLTOS : é um dos motivos do C não ser usado em segurança, pois quando se dá delete em um bloco de memória e temos ainda um ponteiro apontando pra lá podemos ter problemas.

Os ponteiros em C/C++ podem apontar para qualquer endereço de memória e são extremamente flexíveis.

Tipo referencia:

**6.2 Referências**

Em java tudo é referencia. Semelhante aos ponteiros.

Representação de ponteiros: em C/C++ são armazenadas em células de memória. Solução para o problema dos ponteiros soltos: extremamente custosa e não implementada em nenhuma lp. Ou podemos solucionar usando dispose.

Gerenciamento da Heap:

Abordagens para a coleta de lixo:

|  |
| --- |
| * Uso de contador de referências. * Uso de marcar varrer(Abordagem lazy) |

**VERIFICAÇÃO DE TIPOS**

É a atividade responsável por garantir que os operandos de um operador são compatíveis. Um tipo compatível é aquele que para um interpretador/compilador é convertido para um tipo legal. Essa conversão automática é chamada de coerção.

Um erro de tipo é a aplicação de um operador a um operando de um tipo não apropriado;

Uma linguagem é fortemente tipada se os erros de tipo são sempre detectados.