Programação Estruturada
Tipos de Dados

Paradigmas de Linguagens de Programação

Profa. Heloisa – 1º. Sem. 2019

Tipos de dados

Sebesta, R.W. *Concepts of Programming Languages*. 9a.edição/ Addison-Weley, 2009. Capítulo 6.

- Um tipo de dado define uma coleção de valores de dados e um conjunto de operações pré-definidas nesses dados
- Tipos de dados disponíveis em uma linguagem de programação devem espelhar os objetos no espaço de problema do mundo real.
- É fundamental que as linguagens deem suporte a uma coleção de tipos e estruturas de dados apropriados.

Tipos de dados

- Utilidades do sistema de tipos das linguagens:
  - Detecção de erros (mais prática) – a verificação de tipos é dirigida pelo sistema de tipos da linguagem
  - Assistência para a modularização dos programas – pela verificação de tipos através dos módulos
  - Documentação – a declaração de tipo oferece informações importantes sobre o funcionamento do programa.

Tipos de dados

- Descritores
  - Coleção de atributos de uma variável
  - Se os atributos são todos estáticos, o descritor é usado só pelo compilador; são construídos pelo compilador, usualmente como parte da tabela de símbolos e usados durante a compilação
  - Se os atributos são dinâmicos, parte ou todo descritor deve ser mantido durante a execução; nesse caso, é usado pelo sistema de execução.

Tipos de dados

- Tipos primitivos de dados – São os tipos de dados que não são definidos em termos de outros tipos
  - Numéricos
    - Inteiro
    - Ponto flutuante
    - Complexo
    - Decimal
  - Booleano
  - Caractere

Tipos de dados

- Cadeias de caracteres – tipo de dado em que os valores são sequência de caracteres.
- Ordinais definidos pelo usuário
  - Enumeração – aquele em que todos os valores possíveis são enumerados na definição
    - Exemplo em C#

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

  - Subintervalo (Subrange) – subsequência contínua de um tipo ordinal

Tipos de dados

- Array
  - Agregado homogênio de elementos de dados no qual um elemento individual é identificado por sua posição no agregado, com relação ao primeiro elemento.
  - Elementos são do mesmo tipo.
  - Referência a elementos individuais é feita com expressões envolvendo índices.

Tipos de dados

- Array – questões importantes relacionadas:
  - tipo dos índices
    - geralmente inteiros
    - ADA permite qualquer enumeração
  - Limite inferior dos índices
    - Limite inferior implícito
    - Definido pelo usuário
  - alocação de memória
    - Cinco categorias baseadas na amarração de intervalo de índices e de memória

Tipos de dados

- Categorias baseadas na amarração de intervalo de índices e de memória
- Array estático
  - Intervalo de índices e alocação de memória estáticos – feitos antes da execução
  - Exemplo: arrays em C, C++ declarados com a palavra chave static

```
static int a[10];
```


Tipos de dados

- Categorias baseadas na amarração de intervalo de índices e de memória
- Array dinâmico de pilha fixo
  - Intervalo de índices amarrado estaticamente
  - Alocação de memória dinâmica de pilha (feita na elaboração da declaração)
  - Exemplo: arrays em C, C++ declarados sem a palavra chave static

```
int a[10];
```

Tipos de dados – Categorias baseadas na amarração de intervalo de índices e de memória

- Array dinâmico de pilha
  - Intervalo de índices e alocação de memória dinâmicos de pilha, feitos na elaboração da declaração
  - Exemplo em ADA:

```
Get(List_Len);
declare
   List : array (1..List_len) of Integer;
begin
...
end;
(Usuário fornece o tamanho do array; Array alocado
  quando a execução atinge o bloco declare e
  liberado quando a execução sai do bloco)
```


Tipos de dados – Categorias baseadas na amarração de intervalo de índices e de memória

- Array dinâmico de heap fixo
  - Intervalo de índices e alocação de memória dinâmicos, feitos quando o programador solicita, durante a execução do programa.
  - Memória é alocada na heap e não na pilha.
  - Depois de alocados, permanecem fixos.
  - Exemplo em C++:

```
int size;
cout << "Enter size of array: ";
cin >> size;
int *x = new int[size];
for (int i = 0; i < size; i++)
{    //cout << "x[" << i << "] = ";
     x[i] = i + 1; }
```

Tipos de dados – Categorias baseadas na amarração de intervalo de índices e de memória

- Array dinâmico de heap
  - Intervalo de índices e alocação de memória dinâmicos e podem mudar durante a execução do programa
  - Vantagem: flexibilidade: array pode aumentar e diminuir de tamanho
  - Exemplo de classe em C# :

```
ArrayList intList = new ArrayList();
// cria  um objeto da classe sem elementos

intList.Add(nextOne);
// adiciona elementos ao objeto criado
```


Tipos de dados

- Record
  - É um agregado de dados em que os elementos são identificados por nomes
  - Coleção de dados que não tem o mesmo tipo (exemplo: dados sobre um estudante).
- Union
  - Variáveis podem conter dados de tipos diferentes em momento diferentes da execução.
  - Problema: se o tipo union não incluir um indicador de tipo, a verificação de tipo não pode ser feita durante a compilação.

Tipos de dados

- Ponteiros
  - Tipo de dados que contém um intervalo de valores que são endereços de memória e um valor especial, **nil**.
  - Um ponteiro pode ser usado para acessar uma posição na área em que a memória é alocada dinamicamente (heap).
  - Exemplo em C:

```
int *ptr;
int count, init;
...
ptr = &init;
//& operador que produz o endereço de uma variável
count = *ptr;
//* operador que faz a operação de desreferência
```


Tipos de dados

- Ponteiros
  - Aritmética de ponteiros é possível de forma restrita.
    Se ptr é um ponteiro, a expressão `ptr + index` é válida
  - Exemplo em C e C++:

```
int list[10];
int *ptr;
ptr = list; // atribui o endereço de list[0] para ptr
```

  - nome de array sem subscrito é o endereço base do array
  - `*(ptr + 1)` é equivalente a `list[1]`
  - `*(ptr + index)` é equivalente a `list[index]`
  - `ptr[index]` é equivalente a `list[index]`

Tipos de dados

- Referência
  - Semelhante a ponteiro, mas se refere a um dado e não a um endereço.
  - Não há aritmética de referências.
  - C++ inclui um tipo de variável de referência usada para parâmetros formais em definições de funções.
  - É um ponteiro constante que é sempre implicitamente derreferenciado
  - Deve ser inicializado com o endereço de alguma variável em sua definição
  - Depois da inicialização não pode fazer referência a outra variável.
  - São especificadas com o símbolo & precedendo o nome

```
int result = 0;
int &ref_result = result; \\ result e ref_result são sinônimos
```


Verificação de Tipo

- Atividade que certifica que os operandos de um operador são de tipos compatíveis.
- São considerados operadores e operandos:
  - operadores usuais (aritméticos, relacionais, etc)

    Y + Z –3;
  - subprogramas (operadores) e parâmetros (operandos)

    int f(int x) { ….}
    int main() { int a = 10; int b; b = f(a);
  - atribuição (operador) e variavel / expressão (operandos)

    X = Y + Z;

Verificação de Tipo

- Os tipos de operandos são compatíveis com um operador se:
  - são aceitos pelo operador ou
  - podem ser convertidos implicitamente pelo compilador (coerção)
- **Erro de tipo**: aplicação de um operador a um operando de tipo não apropriado.

Verificação de tipo estática

- Feita antes da execução
- Amarração de tipo estática permite quase sempre a verificação de tipo estática (em tempo de compilação)
- Vantagem da detecção de erros em tempo de compilação: quanto antes e erro for encontrado, menor o custo.
- Desvantagem: redução da flexibilidade para o programador

Verificação de tipo dinâmica

- Feita durante a execução do programa.
- Amarração de tipo dinâmica exige verificação de tipo dinâmica.
- Amarração de tipo dinâmica – o tipo da variável é definido durante a execução, por um comando de atribuição
- Exemplo: APL
  List <- 10.5 5.3 4.1 0.0 (list é um array do tipo float)
  List <- 38 (list é um inteiro escalar)

Unions, record variante, equivalence

- A verificação de tipo dinâmica também deve ser feita quando a mesma posição de memória pode armazenar valores de tipos diferentes em momentos diferentes durante a execução
- Ada e Pascal : record variante
- FORTRAN: EQUIVALENCE
- C, C++ : unions

Unions

- Uma union (união) permite criar variáveis capazes de suportar diferentes tipos de dados, no mesmo espaço de memória em momentos diferentes.
- A declaração de uma union é similar à declaração de uma estrutura.
- Com uma union só é alocado espaço para o maior dos objectos que a compõem

```
union Valor{
int ivalor;
double dvalor;
char cvalor;
}val;
```

- significa que *val* poderá armazenar ou um inteiro, ou um double, ou um char.

Tipagem forte

Uma linguagem é considerada ***fortemente tipada*** se erros de tipo podem sempre ser detectados.

- Isso requer que os tipos dos operandos possam sempre ser determinados, seja em tempo de compilação ou em tempo de execução
- Uma linguagem fortemente tipada deve permitir também a detecção, em tempo de execução, de uso de valores de tipo incorreto de variáveis que podem armazenar valores de mais de um tipo.

Tipagem forte

- FORTRAN 95 não é fortemente tipada
  - EQUIVALENCE permite acesso a mesma posição de memória por variáveis de tipos diferentes
- Pascal não é fortemente tipada
  - record variante permite omissão do tag que armazena o tipo corrente de uma variável
- C, C++ não são fortemente tipadas
  - estruturas do tipo union não são checadas

- Ada é **quase** fortemente tipada
  - Permite que o programador suspenda as regras de verificação de tipos solicitando que a verificação não seja feita para um tipo particular de conversão
- Java e C# são **quase** fortemente tipadas no mesmo sentido que ADA
  - permitem conversão explicita de tipo, que pode resultar em erro de tipo

Conversão de tipo

- Conversão de tipo implicita (coerção) – iniciada pelo compilador
- Conversão de tipo explícita (casting) – solicitada pelo programador
- As regras de coerção de uma linguagem tem efeito no valor da verificação de tipo.
- O valor da tipagem forte é enfraquecido pela coerção
- Linguagens que permitem muitas conversões implícitas (Fortran, C, C++) são menos seguras que as que não permitem (Ada, Java, C#).

- Exemplo em JAVA:

```
int a;
float b, c, d;
....
d = b * a;
```

Java permite expressões de modo misto, assim é feita coerção nesse caso e a é convertido para float.

Supondo que houve um erro de digitação e era para ser c no lugar de a, esse erro não é detectado

- Exemplo em ADA:

```
A : Integer;
B, C, D : Float;
....
C := B * A;
```

O compilador encontra um erro de tipo, pois Float e Integer são tipos que não podem ser misturados no operador *

Ada permite poucos casos de conversão implícita, para melhorar a verificação de erros