RESUMO LP PROVA 1

Criado	@17 de março de 2024 15:06
Revisado	☐

VALORES

O que é um valor?

Um valor é definido como qualquer entidade que existe durante a execução de um programa. Isso inclui qualquer coisa que pode ser avaliada, armazenada, ou passada como parâmetro para uma função. Essa definição abrange uma ampla gama de dados, desde números e strings até estruturas mais complexas.

Por que estudar valores?

Os dados representam a matéria-prima da computação. Sem eles, os programas não teriam uma base sobre a qual operar ou manipular informações. Entender os valores é fundamental para compreender como os programas funcionam e como eles podem ser usados para resolver problemas.

Como estudar valores?

Os valores são estudados agrupando-os em tipos. Os tipos são especificações que determinam a classe de valores que podem ser associados a uma variável, além das operações permitidas sobre esses valores. As linguagens de programação (LPs) geralmente oferecem um conjunto de tipos primitivos e fornecem mecanismos para construir tipos compostos. Alguns permitem até tipos recursivos.

Tipos primitivos

Os tipos primitivos são os blocos de construção básicos dos tipos de dados e incluem:

- Lógico (booleans: verdadeiro ou falso)
- Inteiro (números inteiros)
- Real (números fracionários)

- Caractere (caracteres individuais)

Esses tipos são definidos pela implementação da linguagem de programação e são atômicos, ou seja, não são decompostos em tipos mais simples.

Tipos compostos

Os tipos compostos são estruturas de dados que combinam um ou mais tipos primitivos ou outros tipos compostos. Eles incluem estruturas como tuplas, registros, uniões, arranjos, listas, entre outros. Essas estruturas permitem representar dados mais complexos de maneira organizada.

Mecanismos básicos de estruturação

1. **Produto Cartesiano**: Combina conjuntos de valores em n-tuplas ordenadas.
2. **União Discriminada**: Permite estruturas alternativas dentro de um tipo, essencial para criar registros variantes e uniões.
3. **Mapeamento**: Representa funções de um conjunto finito de valores em valores de outro tipo, como arranjos.
4. **Sequência**: Representa um número arbitrário de ocorrências de itens de um determinado tipo.
5. **Conjunto Potência**: Representa todos os possíveis subconjuntos de um conjunto de elementos, permitindo operações de conjuntos típicas.
6. **Recursão**: Permite que um tipo tenha componentes do mesmo tipo, facilitando a representação de estruturas de dados complexas e potencialmente infinitas, como árvores.

Sistema de tipos

O sistema de tipos de uma linguagem de programação pode ser estático ou dinâmico, influenciando como os tipos são verificados. A tipagem estática verifica os tipos em tempo de compilação, enquanto a tipagem dinâmica faz isso em tempo de execução. Cada abordagem tem suas vantagens e desvantagens em termos de segurança e flexibilidade.

Expressões

As expressões são frases de programa que produzem um valor. Podem ser simples, como literais que representam valores fixos, ou complexas, envolvendo operações entre variáveis, chamadas de função, expressões

condicionais, e mais. Elas são fundamentais para a manipulação de dados em programas.

1. Produto Cartesiano

- **O que é:** Combinação de todos os elementos de dois ou mais conjuntos, formando pares, trios, etc., de maneira ordenada, onde cada elemento de um conjunto é pareado com cada elemento do outro conjunto.
- **Exemplo Prático:** Se temos conjuntos de cores `{vermelho, azul}` e de formas `{círculo, quadrado}`, o produto cartesiano nos dá todas as combinações possíveis: `(vermelho, círculo)`, `(vermelho, quadrado)`, `(azul, círculo)`, e `(azul, quadrado)`.

2. União Discriminada

- **O que é:** Estrutura que permite armazenar diferentes tipos de dados em uma mesma variável, mas apenas um tipo de dado por vez, discriminado por uma "tag" ou indicador.
- **Exemplo Prático:** Em um sistema de registro de pets, um registro pode ter uma "tag" que indica se é um `cachorro` (com idade e raça) ou um `peixe` (com tipo de água e temperatura ideal).

3. Mapeamento

- **O que é:** Relaciona chaves a valores, permitindo encontrar rapidamente um valor baseado em uma chave. Cada chave é única e aponta para um valor específico.
- **Exemplo Prático:** Um dicionário onde a chave é a palavra (ex.: "casa") e o valor é o significado da palavra.

4. Sequência

- **O que é:** Coleção de itens ordenados onde a posição de cada item importa. Pode ser infinita e permite operações como concatenação e obtenção de sub-sequências.
- **Exemplo Prático:** Uma lista de compras escrita em um papel, onde você tem `"leite, ovos, pão"`, e a ordem pode determinar a ordem que você pega

os itens na loja.

5. Conjunto Potência

- **O que é:** Conjunto de todos os subconjuntos possíveis de um conjunto dado, incluindo o conjunto vazio e o próprio conjunto.
- **Exemplo Prático:** Para o conjunto `{1, 2}`, o conjunto potência é `{{}, {1}, {2}, {1, 2}}`.

6. Recursão

- **O que é:** Um tipo de dados ou função que se define em termos de si mesmo, permitindo estruturas de dados que podem conter instâncias de si mesmas ou funções que chamam a si mesmas.
- **Exemplo Prático:** Uma estrutura de árvore genealógica, onde cada pessoa pode ter pais e filhos, e esta relação se repete por várias gerações.

Expressões:

A) Literal

- **Definição:** Uma expressão literal representa um valor fixo no código. É a forma mais simples de expressão, pois não requer nenhuma computação para determinar seu valor — o valor já está explicitamente presente no código.
- **Exemplos em Pascal:**
  - Números inteiros: `1234`
  - Números reais: `1.5`
  - Caracteres: `'b'`

B) Agregado

- **Definição:** Expressões agregadas são usadas para construir valores compostos a partir dos valores de seus componentes individuais. Elas permitem agrupar múltiplos valores em uma única entidade.
- **Exemplos:**

- **ML:** `(a*2.0, b/2.0)` cria uma tupla com dois elementos, onde o primeiro é o resultado de `a*2.0` e o segundo é `b/2.0`.
- **Ada:** `anonovo := (y => ano + 1, m => jan, d => 1);` cria um registro com três campos (`y`, `m`, `d`) atribuídos a valores específicos.
- **Pascal:** Não permite a criação direta de agregados para arranjos ou registros através de uma única expressão, exigindo atribuição individual a cada componente.

C) Chamada de Função

- **Definição:** Uma chamada de função calcula um resultado através da aplicação de uma função a um ou mais argumentos. Funciona como a execução de uma série de instruções definidas previamente que retornam um valor.
- **Exemplos:**
  - **ML:** `(if cond then sin else cos) (x)` avalia `cond`, e dependendo do resultado, aplica `sin` ou `cos` a `x`.
  - **C:** `f(x)` chama a função `f` com `x` como argumento.
  - **Observação:** Operadores podem ser vistos como funções especiais que tomam argumentos (operandos) e retornam um resultado, e algumas linguagens permitem que operadores sejam definidos da mesma forma que funções.

D) Expressão Condicional

- **Definição:** Expressões condicionais possuem múltiplas subexpressões, mas apenas uma delas é escolhida e avaliada com base em uma condição. É um meio de executar diferentes códigos dependendo de uma condição booleana.
- **Exemplos:**
  - **C:** `(a > b) ? a : b` retorna `a` se `a` for maior que `b`, caso contrário retorna `b`.
  - **ML:** `if a > b then a else b` funciona de maneira semelhante ao exemplo em C, escolhendo `a` ou `b` baseado na comparação.

E) Acesso a Variáveis e Constantes

- **Definição:** Esse tipo de expressão produz os valores associados a variáveis ou constantes específicas quando seus identificadores são referenciados no código. É a maneira de ler os dados armazenados em locais de memória nomeados.
- **Exemplo:** Se `x` é uma variável com valor `5`, então a expressão `x` produzirá `5`. Se `PI` é uma constante definida como `3.14`, então a expressão `PI` produzirá `3.14`.

Esses tipos de expressões são fundamentais na construção de programas, permitindo representar e manipular dados de diversas formas, conduzindo a uma ampla gama de funcionalidades e comportamentos em software.

ARMAZENAMENTO

Variáveis

- **Conceito:** Objetos que contêm valores, modelando entidades do mundo real com estado mutável.
- **Tempo de Vida:** Geralmente breve, exceto em casos como arquivos, que persistem por mais tempo.
- **Matemática vs. Programação:** Diferente da matemática, onde variáveis têm valores imutáveis, na programação, é comum alterar o valor de uma variável (ex.: `x := x + 1`).

Variáveis Compostas

- **Definição:** Consistem de componentes variáveis, cujos conteúdos podem ser inspecionados e atualizados.
- **Atualização:** Pode ser total (todo o valor) ou seletiva (apenas componentes específicos).

Variáveis de Arranjo

- **Tipos:**
    1. **Estáticos:** Tamanho definido em tempo de compilação.

2. **Dinâmicos:** Tamanho estabelecido na criação da variável.
3. **Flexíveis:** Tamanho e conteúdo podem mudar durante a execução.

Armazenáveis

- **Definição:** Valores que podem ser armazenados em células e não atualizados seletivamente.
- **Exemplo:** Valores primitivos em Pascal, como números e ponteiros.

Tempo de Vida

- **Descrição:** Período entre a criação e a destruição de uma variável, crucial para a gestão de memória.

Variáveis Locais e Globais

- **Locais:** Usadas apenas dentro do bloco onde são declaradas.
- **Globais:** Acessíveis em qualquer bloco, mesmo não sendo declaradas localmente.

Variáveis de Heap

- **Características:** Criadas e destruídas dinamicamente, acessadas via ponteiros, sem um tempo de vida padrão.

Variáveis Persistentes vs. Transientes

- **Persistentes:** Sobrevivem além da execução do programa que as criou (ex.: arquivos).
- **Transientes:** Existem apenas durante a execução do programa.

Referências "Penduradas"

- **Problema:** Ponteiros que apontam para memória desalocada, podendo ocorrer quando variáveis locais são referenciadas fora do seu escopo.

Comandos

- **Função:** Executar ações para atualizar variáveis, caracterizando as linguagens de programação imperativas.

A. Saltos (Skip ou Dummy)

- **Conceito:** O comando de salto é o mais simples, não realizando nenhuma ação. É útil para estruturas condicionais onde uma das ações é não fazer nada.
- **Exemplo:** `if E then C else skip;` — Executa `C` se `E` é verdadeiro, caso contrário, não faz nada.

B. Atribuições

- **Conceito:** Altera o valor de uma variável (`V`) para um novo valor (`E`). É uma das operações mais comuns em programação.
- **Variações:**
  - **Múltipla:** `V1 := ... := Vn := E;` — Atribui `E` a múltiplas variáveis.
  - **Simultânea:** `V1, ..., Vn := E1, ..., En;` — Atribui cada `E` a sua respectiva `V` simultaneamente.
  - **Com Operador Binário:** `V += E;` — Modifica `V` pela operação definida.

C. Chamadas de Procedimento

- **Conceito:** Executa um conjunto de instruções definidas em um procedimento com determinados argumentos.
- **Forma Geral:** `P(AP1, ..., APn);` — `P` é o procedimento aplicado aos argumentos `AP1` a `APn`.

D. Comandos Sequenciais

- **Conceito:** Executa uma série de comandos um após o outro, na ordem em que são escritos.
- **Forma Geral:** `C1; C2; ... ;Cn;` — `C1` é executado antes de `C2`, e assim por diante.

E. Comandos Colaterais

- **Conceito:** Comandos executados sem uma ordem específica, podendo ser não-determinísticos se afetarem variáveis compartilhadas.
- **Exemplo:** `x := 5, x := x + 1;` — A ordem de execução não é definida.

F. Comandos Condicionais

- **Conceito:** Executa subcomandos com base em condições. Apenas um dos subcomandos é escolhido para execução.
- **Exemplos:**
  - Determinístico: `if E then C1 else C2;` — Executa `C1` se `E` é verdadeiro, senão executa `C2`.
  - Não-determinístico: Pode ser útil em programação concorrente, onde a escolha pode depender de condições de execução.

G. Comandos Iterativos (Loops)

- **Conceito:** Executa um subcomando repetidamente, baseado em uma condição de continuação ou em um número definido de vezes.
- **Tipos:**
  - **Indefinidos:** `while E do C;` — Repete `C` enquanto `E` é verdadeiro.
  - **Definidos:** `for V := start to end do C;` — Repete `C` com `V` variando de `start` a `end`.

Expressões com Efeitos Colaterais

- **Conceito:** Expressões cuja avaliação altera o estado do programa, como atualizar o valor de uma variável.

Linguagens Orientadas para Expressões

- **Descrição:** Eliminam distinções entre expressões e comandos, permitindo que a avaliação de expressões retorne valores e cause efeitos colaterais.

AMARRAÇÕES

Conceitos Gerais

- **Amarrações:** Especificações de atributos de entidades de programa, como variáveis e subprogramas.
- **Amarração Estática vs. Dinâmica:** Estática é definida antes da execução e permanece inalterada; dinâmica é estabelecida e pode ser alterada durante

a execução.

Amarrações e Ambientes

- **Ambiente:** Conjunto de amarrações ativas num determinado ponto do programa.
- **Declarações:** Produzem amarrações, associando identificadores a entidades.

Denotáveis

- **Entidades Denotáveis:** Incluem valores primitivos, variáveis, procedimentos/funções e tipos.
- **Amarrações em Pascal:** Definições de constantes, declarações de variáveis, procedimentos, funções e tipos.

Escopo e Estruturas de Bloco

- **Escopo:** Parte do programa onde uma declaração é válida.
- **Estruturas de Bloco:** Podem ser monolíticas (programa inteiro como um bloco), niveladas (programa dividido em blocos distintos) ou aninhadas (blocos dentro de blocos).

Amarração Estática vs. Dinâmica

- **Amarração Estática:** Associada ao ambiente de definição da entidade.
- **Amarração Dinâmica:** Associada ao ambiente de chamada, sendo resolvida em tempo de execução.

Declarações e Definições

- **Declarações:** Frases de programa que elaboram amarrações. Podem ser explícitas ou implícitas.
- **Definições:** Tipos de declarações que produzem amarrações, como a definição de constantes ou a amarração de identificadores a abstrações de procedimentos/funções.

Tipos de Declarações

- **Declarações de Tipos:** Podem definir novos tipos ou amarrar identificadores a tipos existentes.

- **Declarações de Variáveis:** Criam novas variáveis ou amarram identificadores a variáveis existentes.

Declarações Colaterais e Sequenciais

- **Colaterais:** Elaboram subdeclarações independentemente e combinam amarrações.
- **Sequenciais:** Elaboram uma subdeclaração seguida da outra, permitindo o uso de amarrações anteriores em declarações subsequentes.

Declarações Recursivas

- **Definição:** Utilizam amarrações que elas mesmas produzem, suportadas por muitas linguagens modernas, especialmente para tipos, procedimentos e funções.

Blocos

- **Comandos de Bloco:** Contêm declarações locais usadas para executar o comando.
- **Expressões de Bloco:** Contêm declarações que são usadas apenas para avaliar a expressão.

Princípio de Qualificação e Encapsulamento

- **Princípio de Qualificação:** Permite a inclusão de blocos em qualquer classe sintática que especifique algum tipo de computação.
- **Encapsulamento:** Limita a visibilidade de amarrações fora do módulo, mantendo a interface do módulo enxuta.