Tipos de Dados

Paradigmas de Programação – BCC/UFRPE
Lucas Albertins – lucas.albertins@deinfo.ufrpe.br

Agenda

- Introdução
- Tipos Primitivos
- Cadeias de Caracteres
- Tipos Ordinais
- Vetores
- Registros
- Tuplas
- Listas
- Ponteiros
- Referências
- Checagem de Tipos
- Teoria e Tipos de Dados

Introdução

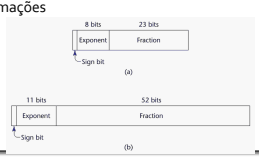
- Um tipo de dados define uma coleção de valores e um conjunto pré-definido de operações sobre tais valores
- Tipos de Dados fornecem:
  - Detecção de Erros (checagem de tipos)
  - Modularização (tipos como interface entre módulos)
  - Documentação
- Um dos passos para se entender uma linguagem é entender seu sistema de tipos
- Tipos primitivos, compostos, tipo abstrato de dados?

Tipos Primitivos

- Não são definidos em termos de outros tipos de dados
- Quase todas linguagens de programação fornecem um conjunto de tipos primitivos
- Alguns são abstrações sobre o hardware
- Sub categorias
  - Numéricos
  - Booleanos
  - Caracteres

Tipos Primitivos - Numéricos

- Inteiros
  - Mapeamento trivial do hardware
    - Ex: 0100 = +4
  - Ex. em Java (bytes): **byte (1), short (2), int(4), long (8)**
- Ponto Flutuante
  - Números reais com aproximações
  - Ex. **float (4), double (8)**



Tipos Primitivos - Numéricos

```
#include <stdio.h>

void main(){

        float a = 1/10.0;

        printf( "%.10f\n",a);

        float pi= 3.14159265358979323846264338327795;

        double piDouble= 3.14159265358979323846264338327795;

        printf("Valor de pi %.10f\n", pi);

        printf("Valor de pi mais preciso %.10f\n", piDouble);

}
```

Tipos Primitivos - Numéricos

- Decimal
  - Armazenam um número fixo de dígitos decimais
  - Vantagem: precisão
  - Desvantagem: Representação em memória é custosa
    - 4 bits por dígito decimal
    - 6 casas decimais = 24 bits
  - Geralmente utilizados para aplicações comerciais (`$$$`)
    - COBOL
    - C#, Java, etc

Tipos Primitivos - Booleanos

- Tipo primitivo mais simples de todos
- Dois valores: Verdadeiro (true), Falso (false)
- Alguns tipos usam inteiros para representá-lo (ex: C)
  - 0 -> false
  - != 0 -> true
- Poderiam ser representados por um bit, mas geralmente se usa-se bytes

Tipos Primitivos - Caracteres

- Codificações numéricas
- ASCII (8 bits) = 128 caracteres
- ISO 8859-1 (8 bits) = 256 caracteres
- Unicode (16 bits)
- UTF-32

Cadeias de Caracteres - Strings

- Sequências de caracteres
- Decisões de design:
  - É um tipo primitivo ou apenas um tipo especial de vetor?
  - O tamanho da cadeia é estático ou dinâmico?
- Operações comuns:
  - Atribuição, cópia
  - Comparação
  - Concatenação
  - Casamento de padrão
  - Substring

Cadeias de Caracteres - Strings

```
#include <stdio.h>
#include <string.h>
Void main()
{
      char a[10]= "teste";
      char b[10]="aaaa";
      b=a; //?
      strcpy(b,a);
      printf("%s",b);
}
```

Cadeias de Caracteres - Strings

- C and C++
  - Não é primitivo
  - Usa `char` arrays e uma biblioteca de funções que fornecem operações
  - Tamanho dinâmico limitado ('\0')
- Fortran and Python
  - Tipo primitivo com atribuição e várias operações
- Java
  - Primitivo ou não?
  - Tamanho estático
- Perl, JavaScript, Ruby, and PHP
  - Fornece mecanismo de casamento de padrão que usa expressões regulares
  - Tamanho dinâmico

Cadeias de Caracteres - Strings

- Python

```
>>> import re
>>> a="abc1234"
>>> b=a
>>>b
'abc1234'
>>> re.findall('[0-9]',a)
['1', '2', '3', '4']
```

Tipos Ordinais

- Enumerações
  - Todos os valores são constantes nomeadas
  - Ex.: em C#
    enum Dias {Seg, Ter, Qua, Qui, Sext, Sab, Dom};
- Subfaixa
  - Subsequência de um tipo ordinal
  - Ex: 12..18 é uma subfaixa do tipo inteiro
  - Ex. Ada:

```
Type Dias is (Seg, Ter, Qua, Qui, Sext, Sab, Dom);
Subtype Semana is Dias range Seg..Sext;
```

- Implementações destes tipos?

Vetores (*Arrays*)

- Agregado homogêneo de elementos onde os indivíduos são identificados pela sua posição em relação ao primeiro elemento
- Algumas decisões:
  - Que tipos podem ser usados para cálculo do índice?
  - O intervalo dos índices é checado?
  - O intervalo dos índices pode ser modificado?
  - Vetores de vetores podem ser construídos?
  - Podem ter seus valores inicializados?

Vetores (*Arrays*)

- Tipos dos índices:
  - FORTRAN, C: integer
  - Java: tipos inteiros
  - Ada: inteiro ou enumeração (incluindo Boolean e char)
- Checagem de intervalo:
  - C, C++, Perl e Fortran não especificam
  - Java, ML, C# especificam
  - Ada – default especifica, mas pode ser desligado

Vetores (*Arrays*) – Vinculações (Bindings)

- Estático – alocação de memória é feita antes da execução (eficiente)
  - C and C++ arrays que incluem o modificador `static` são estáticos
  - Arrays em Java
- Dinâmico – alocação de memória é feita durante a execução (flexível)
  - Classe ArrayList em Java
  - Perl, JavaScript, Python e Ruby têm array dinâmicos

Vetores (*Arrays*) - Inicialização

- Algumas linguagens permitem inicialização no momento da alocação
- C, C++, Java e C#
  - `int list [] = {4, 5, 7, 83};`
- Strings em C e C++
  - `char name [] = "freddie";`
- Inicialização de objetos String em Java
  - `String[] names = {"Bob", "Jake", "Joe"};`

Vetores (*Arrays*) - Operações

- Atribuição
- Concatenação
- Existência de elemento
- Reversão de vetores

Vetores Associativos (Mapas)

- Coleção não ordenada de elementos de dados indexada por um número igual de chaves

```
>>>dic={"joe":32222222,"mary":32211111}

>>>dic['joe']

32222222
```

Registros

- Agregados de elementos heterogêneos no qual os indivíduos são identificados por nomes
- Decisão: Qual a sintaxe de referências aos campos do registro?

1. COBOL
field_name OF record_name_1 OF ... OF record_name_n
2. Notação de ponto
record_name_1.record_name_2. ... record_name_n.field_name

Registro em C++

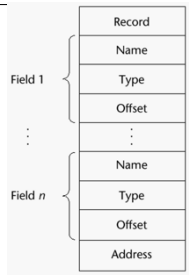
```
#include <iostream>
typedef struct Point {
      int x;
      int y;
      int getNextX() {return x+1;}
      } Point;

int main(){
      Point p;
      p.x = 2;
      std::cout <<p.x << "\n";
      std::cout<<p.getNextX();
      return 1;
}
```

Registro em Ada

```
type Emp_Rec_Type is record
      First: String (1..20);
      Mid: String (1..10);
      Last: String (1..20);
      Hourly_Rate: Float;
  end record;
  Emp_Rec: Emp_Rec_Type;
```

Implementação de Registros

- Um intervalo de endereço (*offset*) relativo ao início do registro é associado com cada campo



Comparação Registros e Vetores

- Registros são usados quando a coleção de valores é heterogênea
- Acesso a elementos de um vetor pode ser muito mais lenta do que o acesso a um campo de um registro quando os índices são dinâmicos

Tuplas

- Similares a registros só que os elementos não são nomeados
- Usados para permitir que funções possam retornar múltiplos valores
- Python: `myTuple = (3, 5.8, 'apple')`
  - Acesso através de índices: `myTuple[1]`
- ML: `val myTuple = (3, 5.8, 'apple');`
  - Acesso primeiro elemento: `#1(myTuple)`

Listas

- Predominantes em linguagens funcionais, hoje em dia já aparecem em linguagens imperativas
- Acesso aos elementos geralmente se dá por funções para acesso a cabeça e cauda da lista
- Lista em ML:
  - Escritas entre colchetes e elementos separados por vírgulas
  - Elementos devem ser do mesmo tipo
  - Função `hd` acessa a cabeça da lista enquanto a função `tl` acessa a cauda da lista

Listas

- Python
  - Listas e Arrays são representados da mesma forma
  - Listas podem ser modificadas
  - Elementos podem ser de qualquer tipo
  - Criando uma lista com uma atribuição:
    `myList = [3, 5.8, "grape"]`
  - Elementos são referenciados com índices começando de zero
    `x = myList[1]` Atribui `5.8` a `x`
  - Compreensão de listas

```
>>>[ x * x for x in range (12) if x % 3== 0 ]
[0, 9, 36, 81]
```

Listas

- Compreensão de Listas em Haskell
  - `[n * n | n <- [1..10]]`
- Ambos C# e Java suportam listas através das classes que implementam a interface List

Uniões

- Estrutura cujos campos compartilham uma área de memória
- Em C são perigosas por falta de verificação de tipos

```
union Data
{
      int i;
      float f;
      char str[20];
};
```

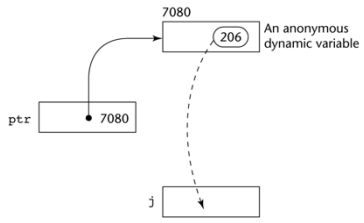
Ponteiros

- Uma variável do tipo ponteiro tem um intervalo de valores que consiste de endereços de memória e um valor especial, nil
- Fornece o poder de endereçamento indireto
- Fornece uma forma de gerenciar a memória dinamicamente
- Um ponteiro pode acessar uma área de armazenamento dinâmica (heap)

Ponteiros

- Decisões de design:
  - Qual é o escopo e tempo de vida de um ponteiro?
  - Qual é o tempo de vida de uma variável na heap?
  - Ponteiros são restringidos pelo tipo de valor para o qual ele aponta?
  - A linguagem suporta tipos ponteiros, tipos de referencia ou ambos?
- Operações: atribuição e desreferenciamento (atribuir o valor do endereço referenciado)
  - j = ptr;
  - j = *ptr;

Ponteiros



- j = *ptr

Problemas com ponteiros

- Ponteiros soltos: endereço para uma variável já liberada

```
void main(){
   float *ptr1, f;
   …
   ptr1 = malloc(sizeof(float));
   …
   free(ptr1);
   …
   f = *ptr1; ⇐ Referência Perdida
}
```

Problemas com ponteiros

- Lixo: uma variável alocada que não está mais acessível

```
void main(){
   int *ptr1, i;
   ...
    for(i = 0; i < 10; i++){
     ptr1 = malloc(sizeof(int)); ⇐ cria lixo
      *ptr1 = i;
   }
  ...
}
```

Referências

- Similar ao ponteiro
- Refere a um objeto ou a um valor em memória e não ao endereço

```
String str;
Str="Uma string literal em Java";
```

- Referência a uma instância da classe String, inicializada como null
- C# tem ambos ponteiros e referências enquanto Java só usa referências

Gerenciamento de Heap

- Um processo complexo em tempo de execução
- Coleta de lixo
  - Contagem de referências: um contador em cada célula que armazena o número de ponteiros que estão apontando para a célula
    - Requer mais espaço, tempo de execução, problemas com células cíclicas
  - Marcar e varrer: aloca as células disponíveis. Depois (1) marca todas como lixo; (2) Pesquisa para saber quais são usadas; (3) Retorna aquelas que são lixo.
    - Causa atrasos no tempo de execução se feito do início. Abordagem melhorada tenta fazer de forma incremental.

Checagem de Tipo

- Garante que os operandos de um operador são de tipos compatíveis (inclusive subprogramas e seus parâmetros)
- Um tipo compatível pode ser permitido para um operador ou é permitido se respeitar as regras da linguagem para ser implicitamente convertido (operação chamada de coerção)
- Um erro de tipo é a aplicação de um operador para um operando de um tipo inapropriado
- Uma linguagem é fortemente tipada se erros de tipo são sempre detectados

Checagem de Tipo

- Se todas as vinculações (bindings) são estáticas, quase todas as checagens de tipo podem ser estáticas
- Se as vinculações (bindings) são dinâmicas, a checagem de tipo deve ser dinâmica
- Linguagens fortemente tipadas
  - ML é fortemente tipada
  - Ada, Java e C# são fortemente tipadas com pequenas exceções
  - C e C++ não são
    - Uniões não são checadas

Teoria e Tipos de Dados

- Teoria de tipos é uma grande área de estudo que envolve matemática, lógica, ciência da computação e filosofia
- Dois ramos
  - Prático: Tipos de dados em linguagens comerciais
  - Abstrato: lambda calculus tipado
- Um sistema de tipo é um conjunto de tipos e as regras que governam seu uso em programas

Teoria e Tipos de Dados

- Um modelo formal de um sistema de tipo é um conjunto de tipos e uma coleção de funções que define as regras do tipo
  - Gramática ou mapeamento de tipos para representar as funções
  - Mapeamentos finitos – modela vetores e funções
  - Produtos cartesianos – modela tuplas e registros
  - Uniões de conjuntos – modela tipos de união
  - Subconjuntos – modela subtipos

Resumo

- Os tipos de dados de uma linguagem são uma grande parte do que determina o estilo da linguagem e sua usabilidade
- Tipos primitivos da maioria das linguagens imperativas incluem tipos numéricos, caractere, Booleanos
- Enumerações e Sub-faixas são convenientes fornecem legibilidade e confiabilidade aos programas
- Vetores e Registros são presentes na maioria das linguagens
- Ponteiros são usados para garantir flexibilidade de endereçamento e para controlar gerenciamento dinâmico de memória

Exercícios

- O que se ganhou e o que se perdeu em Java com a decisão de não fornecer manipulação de ponteiros como em C e C++?
- Por que valores do tipo decimal têm desperdício de memória?
- Analise e escreva uma comparação das funções malloc e free de C com os operadores new e delete de C++.
- Compare as capacidades de manipulação de cadeias de caracteres em C++ e Java
- Para que tipos de A e B a senteça de atribuição A = B é legal em C++ mas não é em Java? E o inverso?

Leitura Adicional

- Capítulo 6 – Tipos de Dados. SEBESTA, R. W. Conceitos de Linguagens de Programação. 9ª ED. BOOKMAN, 2011.
- Próxima aula
  - Capítulo 7 – Expressões e Atribuições. SEBESTA, R. W. Conceitos de Linguagens de Programação. 9ª ED. BOOKMAN, 2011.