Capítulo 6

Tipos de Dados

O que vamos ver?

- Introdução
- Tipos Primitivos
- String
- Enumerações
- Arrays
- Arrays Associativos
- Registros
- Tuplas
- Listas
- Uniões
- Ponteiros e Referências
- Checagem de Tipos
- Tipagem Forte
- Equivalência de Tipos
- Teoria e Tipos de Dados

Introdução

- Tipo de Dado
  - Define os valores e o conjunto de operações permitidos sobre esses valores
- Outros termos importantes:
  - Descritor: a coleção de atributos de uma variável
  - Objeto: uma instância de um tipo de dados definido pelo usuário (tipo abstrato de dados)
- Duas categorias:
  - Primitivos: tipos atômicos, que não são definidos em termos de outros tipos
  - Não primitivos: tipos complexos, que são definidos em termos de outros tipos

Tipos Primitivos

- Não são definidos em termos de outros tipos de dados
- Alguns são meramente “reflexos” do hardware
- Outros necessitam de um pequeno suporte além do hardware para sua implementação

Tipos Primitivos: Inteiros

- Quase sempre são um reflexo exato do hardware, então o mapeamento é trivial
- Podem exister diversos tipos inteiros em uma linguagem, por exemplo:

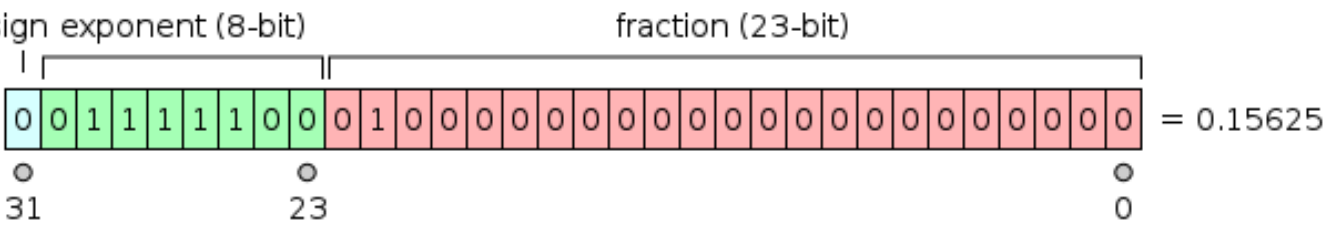
Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Tipos Primitivos: Inteiros

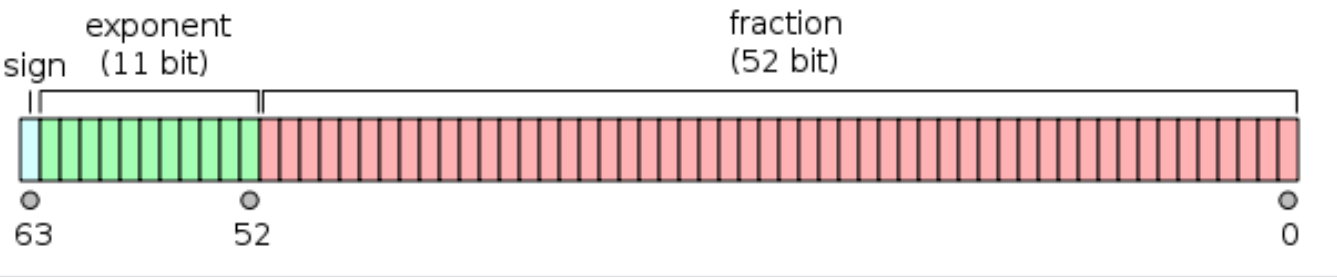
```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <float.h>

int main(int argc, char** argv) {

    printf("CHAR_BIT    :   %d\n", CHAR_BIT);
    printf("CHAR_MAX    :   %d\n", CHAR_MAX);
    printf("CHAR_MIN    :   %d\n", CHAR_MIN);
    printf("INT_MAX     :   %d\n", INT_MAX);
    printf("INT_MIN     :   %d\n", INT_MIN);
    printf("LONG_MAX    :   %ld\n", (long) LONG_MAX);
    printf("LONG_MIN    :   %ld\n", (long) LONG_MIN);
    printf("SCHAR_MAX   :   %d\n", SCHAR_MAX);
    printf("SCHAR_MIN   :   %d\n", SCHAR_MIN);
    printf("SHRT_MAX    :   %d\n", SHRT_MAX);
    printf("SHRT_MIN    :   %d\n", SHRT_MIN);
    printf("UCHAR_MAX   :   %d\n", UCHAR_MAX);
    printf("UINT_MAX    :   %u\n", (unsigned int) UINT_MAX);
    printf("ULONG_MAX   :   %lu\n", (unsigned long) ULONG_MAX);
    printf("USHRT_MAX   :   %d\n", (unsigned short) USHRT_MAX);

    return 0;
}
```

Tipos Primitivos: Ponto Flutuante

- APROXIMAÇÕES para modelar números reais
- Linguagens para uso científico suportam pelo menos dois tipos (**float** e **double**)
- IEEE Floating-Point Standard 754



The number 0.15625 represented as a single-precision IEEE 754-1985 floating-point number. See text for explanation.

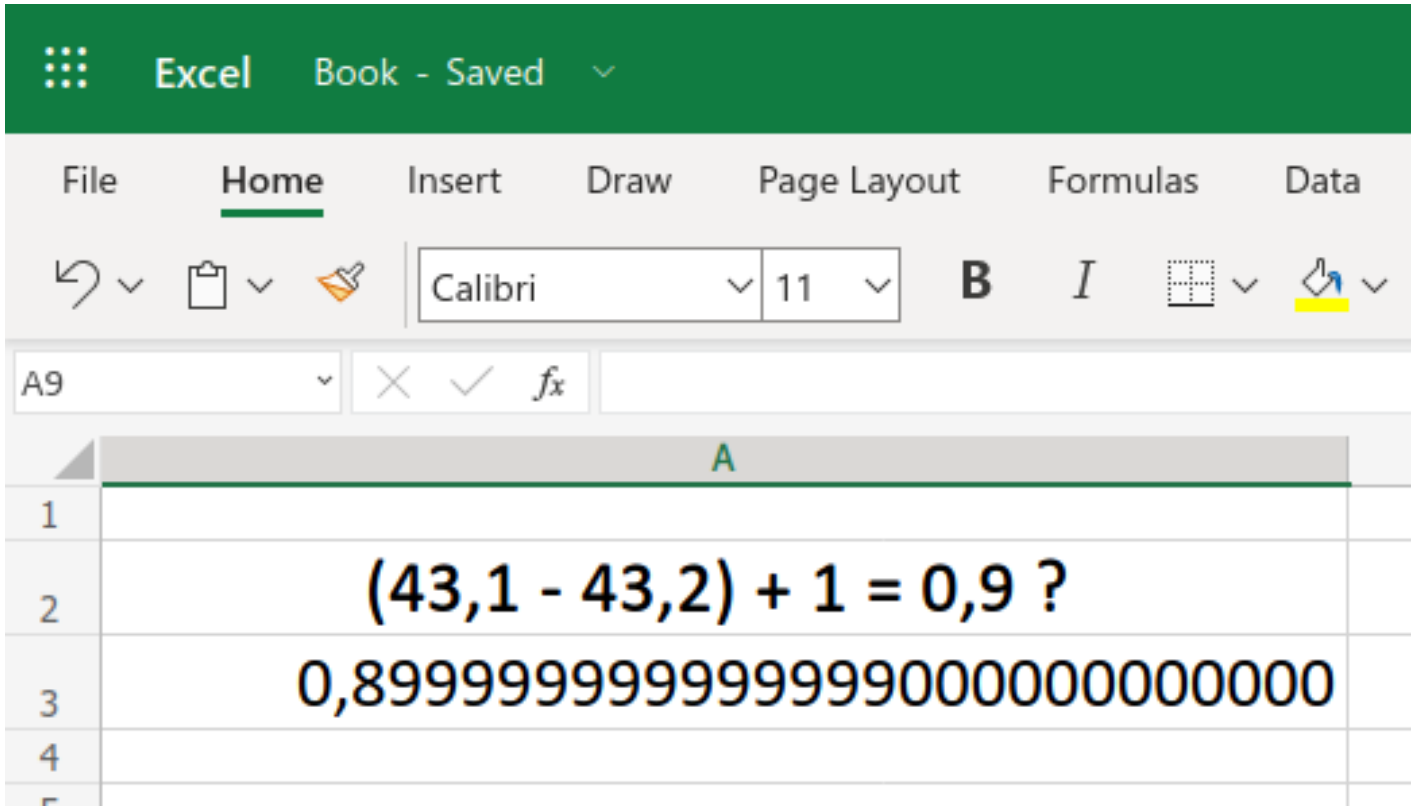


The three fields in a 64bit IEEE 754 float

Tipos Primitivos: Ponto Flutuante

```c
#include <stdio.h>

int main(void){
    float soma = 0.0;

    for (int i = 1; i <= 10000; i++) {
        soma = soma + 0.0001;
    }

    printf("%.15f", soma);

    return 0;
}
```

Tipos Primitivos: Ponto Flutuante



Tipos Primitivos: Ponto Flutuante

- Que conseqüência imediata a imprecisão de armazenamento de pontos flutuantes traz na computação?
  - https://docs.python.org/3/tutorial/floatingpoint.html

```c
#include <stdio.h>

int main(void){
    float x = 1 - 0.1;
    double y = (43.1 - 43.2) + 1;

    int w = 10;
    long int z = 10;

    if (x == y) {
        printf("%s\n", "x = y");
    } else {
        printf("%s\n", "x != y");
    }

    if (w == z) {
        printf("%s\n", "w = z");
    } else {
        printf("%s\n", "w != z");
    }

    return 0;
}
```

Tipos Primitivos: Complexos

- Algumas linguagens têm suporte para o tipo de dados que representa números complexos: C99, Fortran, Lisp, Python
- Cada valor consiste de dois floats:
  - Parte real
  - Parte imaginária
- Em Python, por exemplo: `(7 + 3j)`

```
import cmath

x = cmath.sqrt(-1)
y = cmath.sqrt(-24)
z = cmath.sqrt(9)
print(x, y, z)
```

```
1j 4.898979485566356j (3+0j)
```

Tipos Primitivos: Decimal

- Para aplicações comerciais (dinheiro)
  - Fundamental no COBOL
  - C# tem também
- Armazena um número fixo de dígitos decimais em um formato codificado
  - BCD: binary-coded decimal
- *Vantagem*: precisão
- *Desvantagens*: amplitude limitada, desperdício de memória

Tipos primitivos: Boolean

- Mais simples de todos
- Só 2 valores: 0 ou 1; “true” ou “false”
- Atenção: algumas linguagens entendem “false” como o valor 0, e tudo que não é 0 é considerado “true”
  - Lisp
  - C

Tipos Primitivos: Character

- São armazenados como números inteiros
- Codificação: ASCII
- Codificações alternativas:
  - Unicode-2 (UCS-2)
    - 16 bits
    - Suporta caracteres de quase todas as linguagens
    - Originalmente usado no Java
    - Outras linguagens forneceram suporte
  - Unicode (UCS-4)
    - 32 bits
    - Começou com Fortran, em 2003
  - Outras

Strings

- Valores são seqüências de characters
- Problemas a serem considerados:
  - Deve ser considerado um tipo primitivo ou apenas um tipo especial de array?
  - O comprimento das strings deve ser estático ou dinâmico?

Strings: Operações

- Operações típicas:
  - Alocação
  - Cópia
  - Comparação (=, >, etc.)
  - Concatenação
  - Obtenção de substrings
  - Pattern matching

Strings em Linguagens Selecionadas

- C e C++
  - Não é primitivo
  - Implementada através de arrays de **`char`**
  - Bibliotecas especiais fornecem as operações
- SNOBOL4 (uma linguagem de manipulação de strings)
  - É um primitivo
  - Muitas operações, incluindo pattern matching sofisticado
- Fortran e Python
  - É primitivo
  - Diversas operações
- Java
  - Primitivo através da classe `String`
- Perl, JavaScript, Ruby e PHP
  - Primitivo (?)
  - Têm operações built-in para pattern matching usando expressões regulares

Strings: Tamanho

- **Estático**:
  - COBOL
  - Java (classe String)
- ***Limited Dynamic Length***:
  - C e C++
  - Um caractere especial (“\0”) é utilizado para indicar o final de uma string, ao invés de manter o tamanho
- ***Dynamic*** (sem máximo):
  - SNOBOL4, Perl, JavaScript

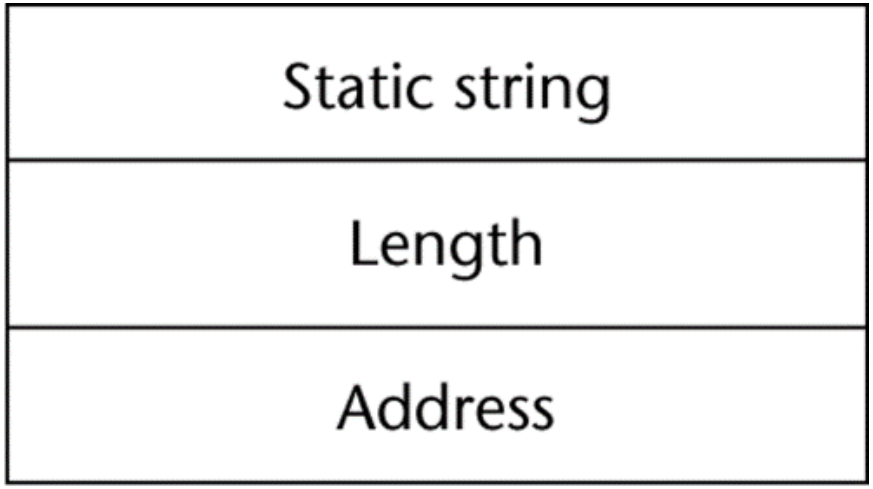
Strings: avaliação de tipo

- Auxílio à escrita de código
- Se é uma linguagem com comprimento estático, é fácil fazer
- Se é uma linguagem com comprimento dinâmico, até que dá para fazer, mas vale a pena o trabalho?

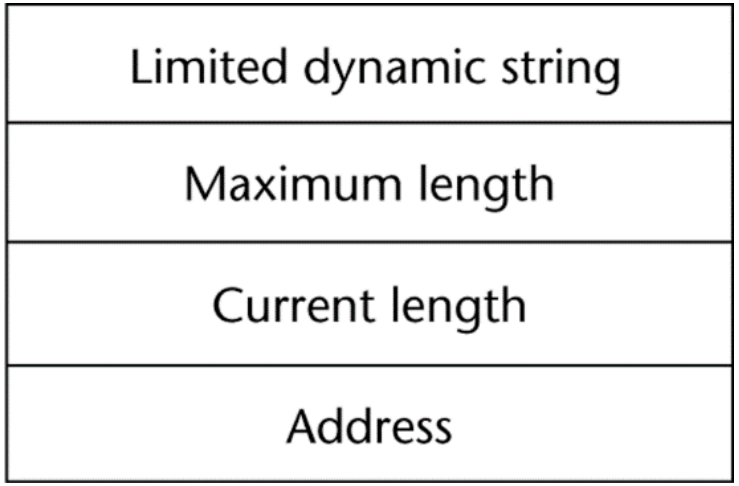
Strings: Implementação

- **Comprimento Estático**:
  - Descritor em tempo de compilação
- **Limited dynamic length**:
  - Podem precisar de um descritor em tempo de execução (exceto em C e C++)
- **Dynamic length**:
  - Precisam de um descritor em tempo de execução
  - Alocação/desalocação é o maior problema de implementação

Strings: Descritores



Descritor em tempo de compilação para strings estáticas



Descritor em tempo de execução para strings limitadas dinamicamente

Até a próxima!

- Material no portal
- Estudem!