Programação Estruturada

Aula 8 - Tipos

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- Até agora nós utilizamos os tipos de dados `int` e `float`
- Nesta aula, vamos estudá-los mais profundamente

Tipos inteiros

- Normalmente, num processador de 32 bits, um `int` é representado usando 32 bits
  - Sendo o primeiro bit responsável por definir o sinal do número
- 01111111111111111111111111111111 é o maior inteiro positivo possível
  - 2.147.483.647 $= (2^{31} - 1)$

Tipos inteiros

- Existe a variação unsigned do tipo `int`, a qual não possui o bit de sinal
  - O `int` que leva em consideração o sinal é chamado de signed
- Nesse caso, o maior inteiro positivo é:
  - 11111111111111111111111111111111
  - 4.294.967.295 $= (2^{32} - 1)$
- Para declarar um `int` como unsigned usamos: `unsigned int`
- O tipo `int` signed é o padrão

Tipos inteiros

- Alguns programas precisam representar números maiores que os suportados pelo `int`
    - Por isso temos o tipo `long`
- Em outros casos, precisamos representar números menores e temos que economizar memória
    - Para isso temos o tipo `short`
- `long` e `short` podem ser signed ou unsigned

Tipos inteiros

- Com isso, podemos especificar um tipo inteiro que satisfaz exatamente o que precisamos:
  - `short int`
  - `unsigned short int`
  - `int`
  - `unsigned int`
  - `long int`
  - `unsigned long int`

Tipos inteiros

- A linguagem C permite abreviações para o uso de `long int` e `short int`
- Basta usar `short` ao invés de `short int` e `long` ao invés de `long int`

Tipos inteiros

- Quais os intervalos de valores para todos esses tipos?
  - Depende do computador e do compilador
- O padrão da linguagem C tem algumas regras:
  - `int` não pode ser representado por menos bits que `short`
  - `long` não pode ser representado por menos bits que `int`
  - Entretanto, pode ser que `short` e `int` tenham a mesma quantidade de bits e/ou `int` e `long` tenham a mesma quantidade de bits

Tipos inteiros

- Valores usuais dos inteiros para um computador de 16 bits

Tipo	Menor valor	Maior valor
short	-32.768	32.767
unsigned short	0	65.535
int	-32.768	32.768
unsigned int	0	65.535
long	-2.147.483.648	2.147.483.647
unsigned long	0	4.294.967.295

Tipos inteiros

- Valores usuais dos inteiros para um computador de 32 bits

Tipo	Menor valor	Maior valor
short	-32.768	32.767
unsigned short	0	65.535
int	-2.147.483.648	2.147.483.647
unsigned int	0	4.294.967.295
long	-2.147.483.648	2.147.483.647
unsigned long	0	4.294.967.295

Tipos inteiros

- Valores usuais dos inteiros para um computador de 64 bits

Tipo	Menor valor	Maior valor
short	-32.768	32.767
unsigned short	0	65.535
int	-2.147.483.648	2.147.483.647
unsigned int	0	4.294.967.295
long	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
unsigned long	0	18.446.744.073.709.551.615

Tipos inteiros

- Para checar os limites do compilador vamos usar o `<limits.h>`

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

int main(void)
{
    printf("short max: %d\n", SHRT_MAX);
    printf("short min: %d\n", SHRT_MIN);

    printf("int max: %d\n", INT_MAX);
    printf("int min: %d\n", INT_MIN);

    printf("long max: %ld\n", LONG_MAX);
    printf("long min: %ld\n", LONG_MIN);

    return 0;
}
```

Tipos inteiros

- Ainda existe o tipo `long long`
- Ele foi adicionado quando surgiram processadores que suportam aritmética de 64 bits
- O tipo `long long` deve ter no mínimo 64 bits

Constantes inteiras

- Constantes inteiras são os números inteiros que aparecem diretamente no código
- Por padrão, escrevemos os números na base decimal
    - Mas podemos escrever em octal ou hexadecimal

Constantes inteiras

- **Decimal:** contém dígitos de 0 até 9 e não pode começar com 0
  - 15, 255, 32767
- **Octal:** contém dígitos de 0 até 7 e devem começar com 0
  - 013, 0377, 077777
- **Hexadecimal:** contém dígitos de 0 até 9 e letras de a até f, eles devem começar com 0x
  - 0xff, 0xff, 0x7a9f
  - As letras podem ser maiúsculas ou minúsculas, não tem diferença

Constantes inteiras

- Números octais e hexadecimais são apenas uma forma diferente de escrever números
- A forma de armazenar continua a mesma

Overflow

- Ao realizar operações aritméticas, o resultado pode ser muito grande para ser representado pelo tipo especificado
    - Se isso acontecer, temos um overflow

```
int x = 2147483647+1;

printf("%d\n", x);
```

Overflow

- Utilizando `unsigned int` nós conseguimos representar números maiores (positivamente)

```
unsigned int x = 2147483647+1;

printf("%u\n", x);
```

Lendo e escrevendo inteiros

- Para ler a escrever tipos unsigned, short ou long, vamos usar outros especificadores de conversão
- Para o `int` nós usamos `%d`, vamos ver os especificadores de conversão para as suas variações

Lendo e escrevendo inteiros

- Para `unsigned int`, temos:
  - `%u` - base decimal
  - `%o` - base octal
  - `%x` - base hexadecimal

Lendo e escrevendo inteiros

- Para `unsigned int`, temos:

```
unsigned int u;

scanf("%u", &u);
printf("%u", u);

scanf("%o", &u);
printf("%o", u);

scanf("%x", &u);
printf("%x", u);
```

Lendo e escrevendo inteiros

- Para `short`, colocamos a letra `h` antes do `d`, `u`, `o` ou `x`

```
short s;

scanf("%hd", &s);
printf("%hd", s);
```

Lendo e escrevendo inteiros

- Para `long`, colocamos a letra `l` antes do `d`, `u`, `o` ou `x`

```
long l;

scanf("%ld", &l);
printf("%ld", l);
```

Lendo e escrevendo inteiros

- Para `long long`, colocamos a letra `ll` antes do `d`, `u`, `o` ou `x`

```
long long ll;

scanf("%lld", &ll);
printf("%lld", ll);
```

Tipos de ponto flutuante

- Quando precisamos representar números fracionários (com vírgula), nós usamos os tipos de ponto flutuante
- C possui três tipos de ponto flutuante
  - float
  - double
  - long double

Tipos de ponto flutuante

- A diferença entre os tipos é a precisão
- O C não especifica o quanto um tipo deve ser mais preciso que o outro, mas sabemos que `long double` é mais preciso que `double` que é mais preciso que `float`

Tipos de ponto flutuante

- A maioria dos computadores utilizam o padrão IEEE 754 para representação de um número de ponto flutuante
- Ele prover dois formatos principais
    - Precisão simples (32-bits)
    - Precisão dupla (64-bits)

Tipos de ponto flutuante

- Os números são representados em três partes
  - Sinal
  - Expoente
  - Fração

Tipos de ponto flutuante

- O número de bits utilizados para o expoente determina o tamanho que o número pode ter
- O número de bits utilizados pela fração determina a precisão do número

Tipos de ponto flutuante

- Para precisão simples
  - O expoente possui 8 bits
  - A fração possui 23

Tipos de ponto flutuante

Tipo	Menor valor positivo	Maior valor	Precisão
float	$1,17549 \times 10^{-38}$	$3,40282 \times 10^{38}$	6 dígitos
double	$2,22507 \times 10^{-308}$	$1,79769 \times 10^{308}$	15 dígitos

Constantes de ponto flutuante

- Constantes de ponto flutuante podem ser escritas de diversas formas
- Todas as constantes abaixo são válidas e representam o número `57.0`:
  - `57.0` `57.` `57.0e0` `57E0` `5.7e1` `5.7e+1` `.57e2` `570.e-1`

Constantes de ponto flutuante

- Uma constante de ponto flutuante precisa ter um ponto e/ou um expoente
- O expoente é especificado pela letra `e` ou `E` seguido de um número
- O expoente indica um número elevado a 10
  - `e5` = 10^{5}
  - `e-5` = 10^{-5}

Constantes de ponto flutuante

- Por padrão, as constantes de ponto flutuante são representadas como `double`
  - Isto não costuma causar problemas para as variáveis do tipo `float` pois o valor da constante é convertido automaticamente quando necessário
- Para forçar uma constante ser do tipo `float` basta colocar um `f` ou `F` no final do número
  - `57.0f`

Lendo e escrevendo números de ponto flutuante

- Já vimos como são usados os especificadores de conversão `%e` , `%f` e `%g` para ler `floats`
- Para o tipo `double` e `long double` fazemos algumas modificações

Lendo e escrevendo números de ponto flutuante

- Para `double`, colocamos a letra `l` antes do `f`, `e` ou `g`:

```
double d;

scanf("%lf", &d);
printf("%lf", d);
```

Lendo e escrevendo números de ponto flutuante

- Para `long double`, colocamos a letra `L` antes do `f`, `e` ou `g`:

```
long double ld;

scanf("%Lf", &ld);
printf("%Lf", ld);
```