# MC-102 — Aula 03
# Variáveis e Atribuições

Instituto de Computação – Unicamp

Segundo Semestre de 2011

# Roteiro

1. Variáveis
2. Constantes
3. Atribuição
4. Atribuição

# Variáveis

**Definição**

Variáveis são locais onde armazenamos valores na memória. Toda variável é caracterizada por um **nome**, que a identifica em um programa, e por um **tipo**, que determina o que pode ser armazenado naquela variável.

- Durante a execução do programa, um pedacinho da memória corresponde a variável

# O endereço de uma variável

- Toda variável tem um endereço de memória associado a ela. Esse endereço é o local onde essa variável é armazenada no sistema (como se fosse o endereço de uma casa, o local onde as pessoas "são armazenadas").
- Normalmente, o endereço das variáveis não são conhecidos quando o programa é escrito.
- O endereço de uma variável é dependente do sistema computacional e também da implementação do compilador C que está sendo usado.
- O endereço de uma mesma variável pode mudar entre diferentes execuções de um mesmo programa C usando uma mesma máquina.

# Declarando uma variável

Declara-se da seguinte forma: **TIPO-variavel NOME-variavel**

- Exemplos Corretos
  - int soma;
  - float div;
  - char resposta;
- Exemplos Incorretos
  - soma int;

# Variáveis inteiras

- Variáveis utilizadas para armazenar valores inteiros, em formato binário.
  Ex: $13_{10} = 1101_2$
- **int**: Inteiro cujo comprimento depende do computador. É o inteiro mais utilizado. Em computadores *Pentium*, ocupa 32 bits e pode armazenar valores de -2.147.483.648 a 2.147.483.647.
- **unsigned int**: Inteiro cujo comprimento depende do computador e que armazena somente valores positivos. Em computadores *Pentium*, ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295.

# Variáveis inteiras

- **long int**: Inteiro que ocupa 32 bits e pode armazenar valores de -2.147.483.648 a 2.147.483.647, independente do computador.
- **unsigned long int**: Inteiro que ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295, independente do computador.
- **short int**: Inteiro que ocupa 16 bits e pode armazenar valores de -32.768 a 32.767.
- **unsigned short int**: Inteiro que ocupa 16 bits e pode armazenar valores de 0 a 65.535.

# Variáveis de tipo caracter

- Variáveis utilizadas para armazenar letras e outros símbolos existentes em textos.
- **char**: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de -128 a 127.
- **unsigned char**: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de 0 a 255.

# Variáveis de tipo ponto flutuante

- Armazenam valores reais, da seguinte forma

$$(-1)^{sinal} \cdot mantissa \cdot 2^{expoente}$$

  Ex: $0.5 = (-1)^0 \cdot 1 \cdot 2^{-1}$
- Para o programador, funciona como se ele armazenasse números na forma decimal.
- Possuem problemas de precisão (arredondamento).
- **float**: Utiliza 32 bits, sendo 1 para o sinal, 8 para o expoente e 23 para a mantissa. Pode armazenar valores de $(+/-)10^{-38}$ a $(+/-)10^{38}$
- **double**: Utiliza 64 bits, sendo 1 para o sinal, 11 para o expoente e 52 para a mantissa. Pode armazenar valores de $(+/-)10^{-308}$ a $(+/-)10^{308}$

# Regras para nomes de variáveis em C

- **Deve** começar com uma letra (maíuscula ou minúscula) ou subcrito(_). **Nunca** pode começar com um número.
- Pode conter letras maíusculas, minúsculas, números e subscrito.
- Não pode-se utilizar como parte do nome de uma variável:

```
{ ( + - * / \ ; . , ?
```

# Regras para nomes de variáveis em C

As seguintes palavras já tem um significado na linguagem C e por esse motivo não podem ser utilizadas como nome de variáveis:

| | | | | |
|---|---|---|---|---|
| `auto` | `double` | `int` | `struct` | `break` |
| `enum` | `register` | `typedef` | `char` | `extern` |
| `return` | `union` | `const` | `float` | `short` |
| `unsigned` | `continue` | `for` | `signed` | `void` |
| `default` | `goto` | `sizeof` | `volatile` | `do` |
| `if` | `static` | `while` | | |

# Constantes

- Constantes são valores previamente determinados e que, por algum motivo, devem aparecer dentro de um programa (veremos adiante onde elas podem ser usadas).
- Assim como as variáveis, as constantes também possuem um tipo. Os tipos permitidos são exatamente os mesmos das variáveis, mais o tipo `string`, que corresponde a uma sequência de caracteres.
- Exemplos de constantes:

```
85, 0.10, 'c', "Hello, world!"
```

# Constantes inteiras

- Um número na forma decimal, como escrito normalmente
  Ex: `10`, `145`, `1000000`
- Um número na forma hexadecimal (base 16), precedido de 0x
  Ex: `0xA` ($0xA_{16} = 10_2$), `0x100` ($0x100_{16} = 256_2$)
- Um número na forma octal (base 8), precedido de 0
  Ex: `010` ($0x10_8 = 8_2$)

# Constantes do tipo de ponto flutuante

- Um número decimal. Para a linguagem C, um número só pode ser considerado um número decimal se tiver uma parte "não inteira", mesmo que essa parte não inteira tenha valor zero. Utilizamos o ponto para separarmos a parte inteira da parte "não inteira".
  Ex: `10.0`, `5.2`, `3569.22565845`
- Um número inteiro ou decimal seguido da letra e e um expoente. Um número escrito dessa forma deve ser interpretado como:

$$numero \cdot 10^{expoente}$$

Ex: 2e2 ($2e2 = 2 \cdot 10^2 = 200.0$)

# Constantes do tipo caracter

- Uma constante do tipo caracter é sempre representado por uma letra entre aspas simples.
  Ex: `'A'`

# Constantes do tipo string

- Uma constante do tipo string é um texto entre aspas duplas
  Ex: `"Hello, world!"`

# Atribuição

Atribuir um valor de uma expressão a uma variável significa calcular o valor daquela expressão e copiar aquele valor para uma determinada variável.

- O comando de atribuição em C é =
- A sintexe é **variável = valor;**
- Exemplos
  - int a;
  - float c;
  - a=5;
  - c=67.8597;

# Atribuição

O comando de atribuição pode conter expressões matemáticas do lado direito

- A sintexe é **variável = Expressão;**
- Atribuir um valor de uma expressão a uma variável significa calcular o valor daquela expressão e copiar aquele valor para uma determinada variável.
- Exemplos
  - int a;
  - float c;
  - a=6+5+8;
  - c=67.8597-9.059686+887576;

# Atribuição

No exemplo abaixo, a variável `soma` recebe o valor calculado da expressão `a + b`

```
soma = a + b;
```

Variável

Expressão

- O operador de atribuição é o sinal de igual ($=$)

À esquerda do operador de atribuição deve existir somente o nome de uma **variável**.

$=$

À direita, deve haver uma **expressão** cujo valor será calculado e armazenado na variável

# Expressões Simples

- Uma constante é uma expressão e como tal, pode ser atribuída a uma variável (ou em qualquer outro lugar onde uma expressão seja necessária).
  Ex: `a = 10;`
- Uma variável é uma expressão.
  Ex: `a = b;`

# Exemplos de atribuição

```
int a,b;
float f,g;
char h;


a = 10;
b = -15;
f = 10.0;
h = 'A';

a = b;
f = a;
```

# Exemplos errado de atribuição

```
int a,b;
float f,g;
char h;
```

```
a b= 10;
f = 10.0
d = 85;
```

# Estrutura de um programa em C

```
// Diretivas do pré-processador
#include <stdio.h>
// Declarações de variáveis globais
int a,b;
main() {
// Declarações de variáveis locais
int n;
float resultado;
// Código do programa principal
printf("Digite um número inteiro para calcular sua raiz qua
scanf("%d", &n);
resultado = n*n;
printf("O valor quadratico do numero %d é %f", n, resultado
return 0;
}
```

# Informações Extras : Tabela ASC

- Variáveis tipo caracter são, na verdade, variáveis inteiras que armazenam um número associado ao símbolo. A principal tabela de símbolos utilizada pelos computadores é a tabela ASCII (*American Standard Code for Information Interchang*), mas existem outras (EBCDIC, Unicode, etc .. ).
- Toda constante do tipo caracter pode ser usada como uma constante do tipo inteiro. Nesse caso, o valor atribuído será o valor daquela letra na tabela ASCII.

# Constantes: Tabela ASCII

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0<br>16 | Caracteres de Controle | | | | | | | | | | | | | | | |
| 32 | | ! | ” | # | $ | % | & | ’ | ( | ) | * | + | , | - | . | / |
| 48 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 64 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 80 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | / | ] | ^ | _ |
| 96 | ‘ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 112 | p | q | r | s | t | u | v | w | x | y | z | { | — | } | ~ | |

# Informações Extras: Obtendo o tamanho de um tipo

O comando `sizeof(tipo)` retorna o tamanho, em bytes, de um determinado tipo. (Um byte corresponde a 8 bits).

**Exemplo**

```
printf ("%d", sizeof(int));
```

Escreve `4` na tela (*Pentium*).

# Informação Extra: Variáveis que guardam endereços

- Armazenam o endereço de outras variáveis.
- Para cada tipo de dados, existe um tipo para guardar o seu endereço, indicado por * antes do nome da variável.
- `int *endereço`: Endereço de uma variável inteira.
- `float *endereço`: Endereço de uma variável de ponto flutuante.
- `char *endereço`: Endereço de uma variável de caracter.
- Estas variáveis são chamadas apontadores.
- O endereço de uma variável também é uma expressão, e é obtido colocando-se o símbolo & antes do nome da variável. Ex: `endereco = &a;`

# Estrutura de um programa em C

```
int *endereco;

endereco = &a;
```