Engenharia de Computação

Fundamentos de Programação

Aula 03 – Variáveis

Prof. Muriel de Souza Godoi
muriel@utfpr.edu.br

Variáveis

- Posição específica de memória, utilizada para armazenar um valor que pode ser modificado;
- Devem ser declaradas antes da utilização:

```
tipo nome_variavel;
```

- Declaração de variáveis pode ser feita:
  - Escopo global :
    - Variáveis globais: declaradas fora das funções
  - Escopo local:
    - Variáveis locais: declaradas dentro das funções ou como parâmetros das funções
- O acesso às variáveis locais tem preferência em relação às variáveis globais

Variáveis - Escopo

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

int global;

int main(){
    int local;

    printf("Hello world!\n");
    return 0;
}//main
```

Área de Variáveis Globais

Área de Variáveis Locais

Variáveis

- Comando de atribuição:
  - Permite fornecer um valor a uma certa variável;
  - Deve ser realizada dentro de uma função;
  - Tipo de informação deve ser compatível com o tipo de variável utilizada;
  - Operador de atribuição: =
    - à esquerda do = será uma variável, conhecido como *lvalue*
    - à direita do = pode ser constantes (ex:número) ou variáveis

```
nome_variavel = valor;
nome_variavel = expressão;
```

- Atribuição pode ser feita na declaração da variável. Exemplo:

```
int idade = 0;
```

Boa pratica para evitar
lixo de memória

Tipos de Variáveis

- Define como os dados armazenados em uma variável serão interpretados

Mais usados na disciplina

Tipo	Bits	Intervalo de valores
char	8	-128 A 127
unsigned char	8	0 A 255
signed char	8	-128 A 127
int	32	-2.147.483.648 A 2.147.483.647
unsigned int	32	0 A 4.294.967.295
signed int	32	-32.768 A 32.767
short int	16	-32.768 A 32.767
unsigned short int	16	0 A 65.535
signed short int	16	-32.768 A 32.767
long int	32	-2.147.483.648 A 2.147.483.647
unsigned long int	32	0 A 4.294.967.295
signed long int	32	-2.147.483.648 A 2.147.483.647
float	32	1,175494E-038 A 3,402823E+038
double	64	2,225074E-308 A 1,797693E+308
long double	96	3,4E-4932 A 3,4E+4932

BACKES, A. Linguagem C: completa e descomplicada

- O padrão C11 (ISO/IEC 9899:2011) possui mais tipos de variáveis
- Deve-se verificar o suporte da compilador (gcc/linux ou mingw-gcc/windows)

Tipos de Variáveis

```
float salario = 1.2345;
char letra = 'a';
short int idade = 22;
double preco = 315.59;
```

- **Considerações**:
  - Linguagem é **case-sensitive** (sensível ao caso): diferencia letras maiúsculas de minúsculas, são diferentes:
    - `Preco, preco, pReco, prEco, preCo, precO, PRECO`
  - Toda instrução termina com ponto-e-vírgula, ou seja ;
  - Por padrão **`float`** e **`double`** usam o . para representar decimal
    - Adotam padrão: 32-bit e 64-bit IEEE 754 de precisão
    - Exemplo: preco = 601.57;

- Números sem . são interpretados por padrão como **`int`**
- Números com . são interpretados por padrão como **`double`**

Palavras Reservadas

Não podem ser utilizadas como nome de variável:

- Palavras que começam com número
- Palavras ou funções reservadas na linguagem C (ver tabela abaixo)

Lista de palavras-chave da linguagem C							
auto	double	int	struct	break	else	long	switch
case	enum	if	typeof	continue	float	return	while
union	const	for	short	unsigned	char	extern	signed
void	default	do	sizeof	volatile	goto	register	static

BACKES, A. Linguagem C: completa e descomplicada

Tipos de Variáveis

- Caso precise representar números muito grandes ou muito pequenos, como fazer ?
  - Usar notação E
  - Exemplo: Constant Planck (usado na Física) que possui valor: 6.62607×10^{-34}, na linguagem C ficaria:

```
double planck = 0;
planck = 6.62607E-34;//seria a representação 6.62607x10-34
```

- E se houver números além da capacidade de um **long double** (> 96bits), ou número complexos ?
  - Existem tipos para número complexos, exemplo: long double _complex (ver manual C99 e C11)
  - Uso de bibliotecas, por exemplo openssl, para números muito grandes, usados em criptografia por exemplo...

Exibindo as Variáveis

```c
int idade;  // Declaração da variável
idade = 18; // Atribuição de valor
printf("Idade: %i \n", idade);
```

O programa irá escrever uma mensagem na tela: `"Idade: 18"`

- Primeiro argumento de printf, que está separado por uma vírgula, é uma string a ser impressa na tela. Os próximos argumentos seguem a ordem dos especificadores de acesso/formato.
- Para exibir o valor de uma variável no que será impresso na tela, precisamos incluir **tags** conhecidas por **especificadores de formato**

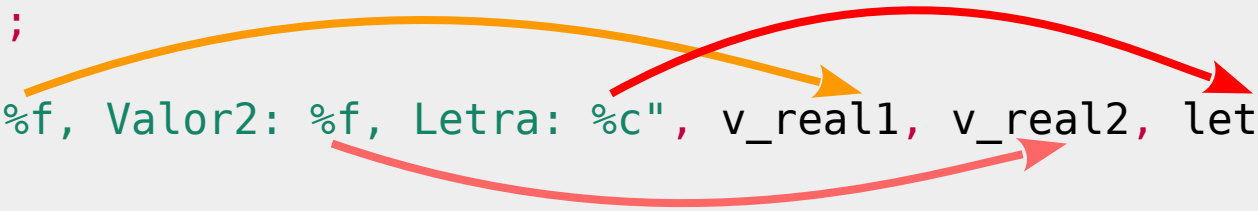
- Especificador de formato: começa com o símbolo `%` e uma **letra**
  - `%i` ou `%d` → exibe um número inteiro
  - `%f` → exibe um número ponto-flutuante
  - `%c` → exibe um caractere

Exibindo Variáveis

- Deve-se **sempre** utilizar o mesmo número de variáveis de acordo com o número de tags
- Exemplo mostrando 3 variáveis:

```
float v_real1 = 5.0;
float v_real2 = 8.678;
char letra = 'a';

printf("Valor1: %f, Valor2: %f, Letra: %c", v_real1, v_real2, letra);
```



- Usos incorretos:

```
printf("Valor1: %f , Valor2: %f , Letra: %c \n", v_real1, v_real2);
printf("Valor1: %f , Valor2: %f \n", v_real1, v_real2, letra);
```

Identificadores de tipo

Na tabela ao lado temos o especificador de formato a ser utilizado na função **printf** para cada tipo de variável

Tipo	Formato para uso com scanf ou printf
char	%c
unsigned char	%c
signed char	%c
int	%i ou %d
unsigned int	%u
signed int	%i ou %d
short int	%hi
unsigned short int	%hu
signed short int	%hi
long int	%li
signed long int	%li
unsigned long int	%lu
float	%f
double	%lf
long double	%Lf
Strings (array de char)	%s

Atribuição em outras bases

- Dependendo da forma como representar um número constante, altera-se a base ou tipo do mesmo
  - **Prefixo**: indica a base numérica
    - **0x***número_hexadecimal* ou **0X***número_hexadecimal*
      - Ex: **0x**2A equivale a 42 base decimal
    - **0***número_octal*
      - Ex: **0**76 equivale a 62 base decimal
    - **0b***número_binário* ou **0B***número_binário*
      - Ex: **0b**100101 equivale a 37 base decimal

Atribuição com modificadores

- **Sulfixo**: representação de tipo a ser usado
  - Número Inteiro:
    - Sem sulfixo: representado como **int**
    - *número***U** ou *número***u** , representa em **unsigned int**
    - *número***L** ou *número***l** , representa em **long int**
    - *número***UL** ou *número***ul** , representa em **unsigned long int**
    - *número***LL** ou *número***ll** , representa em **long long int**
    - *número***ULL** ou *número***ull** , representa em **unsigned long long int**
  - Número com casa decimal
    - Sem sulfixo: representado como **double**
    - *número***f** ou *número***F**, indica **float**
    - *número***l** ou *número***L** , indica **long double**
- **Exemplo:**
  - **0x**15F**UL** , representa o 21 em **unsigned long int**

Exibindo variáveis Formatadas

- Exemplos formatação de saída do printf():

```
printf("%10iFIM\n",-1234); // -1234 alinhado direita e dentro de 10 espaços
printf("%+10iFIM\n",1234); // +indica sinal
printf("%+010iFIM\n",1234); // 0, entre + e 10, indica preencher com 0s
printf("%+-10iFIM\n",1234); //+indica sinal e -10 indica alinhado esquerda
printf("%fFIM\n",1234.112); // padrão 6 casas decimais precisão
printf("%.2fFIM\n",1234.112); //mostrar 2 casas precisão após o .
printf("%+.2fFIM\n",1234.112); //+ indica sinal
printf("%+10.2fFIM\n",1234.112); //alinhamento direita e em 10 espaços
printf("%xFIM\n",60); //0x3c mostrar em hexadecimal
printf("%#xFIM\n",60); //0x3c mostrar em hexadecimal com 0x
```

```
     -1234FIM
     +1234FIM
+000001234FIM
+1234      FIM
1234.112000FIM
1234.11FIM
+1234.11FIM
  +1234.11FIM
3cFIM
0x3cFIM
```

Consulte o manual da função printf para outros tipos e configurações suportadas! **`man printf`**

Tipos e memória ocupada

- Tamanho ocupado: **sizeof**(tipo ou nome_variável);

```c
int a = 1234567;
printf("Um char ocupa %i byte\n", sizeof(char));
printf("Um int ocupa %i bytes\n", sizeof(int));
printf("A variavel \"a\" ocupa %i bytes\n", sizeof(a));
printf("Um float ocupa %i bytes\n", sizeof(float));
printf("Um double ocupa %i bytes\n", sizeof(double));
printf("Um short int ocupa %i bytes\n", sizeof(short int));
printf("Um long int ocupa %i bytes\n", sizeof(long int));
printf("Um long double ocupa %i bytes\n", sizeof(long double));
```

```
Um char ocupa 1 byte
Um int ocupa 4 bytes
A variavel "a" ocupa 4 bytes
Um float ocupa 4 bytes
Um double ocupa 8 bytes
Um short int ocupa 2 bytes
Um long int ocupa 8 bytes
Um long double ocupa 16 bytes
```

Constantes

- Em C é possível definir constantes simbólicas (define)
  - Também denominada **macro**.

- Sintaxe:

```
#define NOME valor
```

- Exemplo:

```
#define PI 3.1416
```

  - O pré-processador substitui o **NOME** pelo **valor**
  - Facilita manuteção de código
  - Variável como constante (**const**)
    - **Sintaxe**: `const tipo nome_var = valor;`
    - **Exemplo**: `const double PI = 3.1415;`

Constantes

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

/* Criação de constantes */

#define VALOR 10
#define PI 3.1416

int main(){
    printf("Valor da constante N: %d \n", VALOR );
    printf("Valor da constante PI: %.4f \n", PI );
}//main
```

Exercícios

- **1)** Elabore um programa que exiba o seu nome na primeira linha e o seu e-mail segunda. Em seguida, exiba uma mensagem solicitando para o usuário pressionar uma tecla. Quando o usuário pressionar, exiba em uma nova linha o nome do seu amigo e, em outra, o e-mail dele.
  - **Dica:** Pesquise a função `getchar` para capturar uma tecla do usuário

- **2)** Declare duas variáveis inteiras A e B e atribua valores diferentes. Em seguida, efetue a troca dos valores de forma que, a variável A passe a possuir o valor da variável B, e que a variável B passe a possuir o valor da variável A. Apresente os valores iniciais e finais de A e B.