BCC 201 - Introdução à Programação

Variáveis, Comandos de Atribuição e Comando de Entrada e Saída

Guillermo Cámara-Chávez
UFOP

Estrutura Básica de um programa C I

```
< diretivas do pré-processador >
< declarações globais >;
int main()
{
    < declarações locais >;  /* comentário */
    < instruções >;
    return 0;
}
< outras funções >
```

Estrutura Básica de um programa C II

```cpp
/* Prog. C++: Bom dia */
#include <iostream>
using namespace std;
int main()
{
    cout<<"Bom Dia!!";
    return 0;
}
```

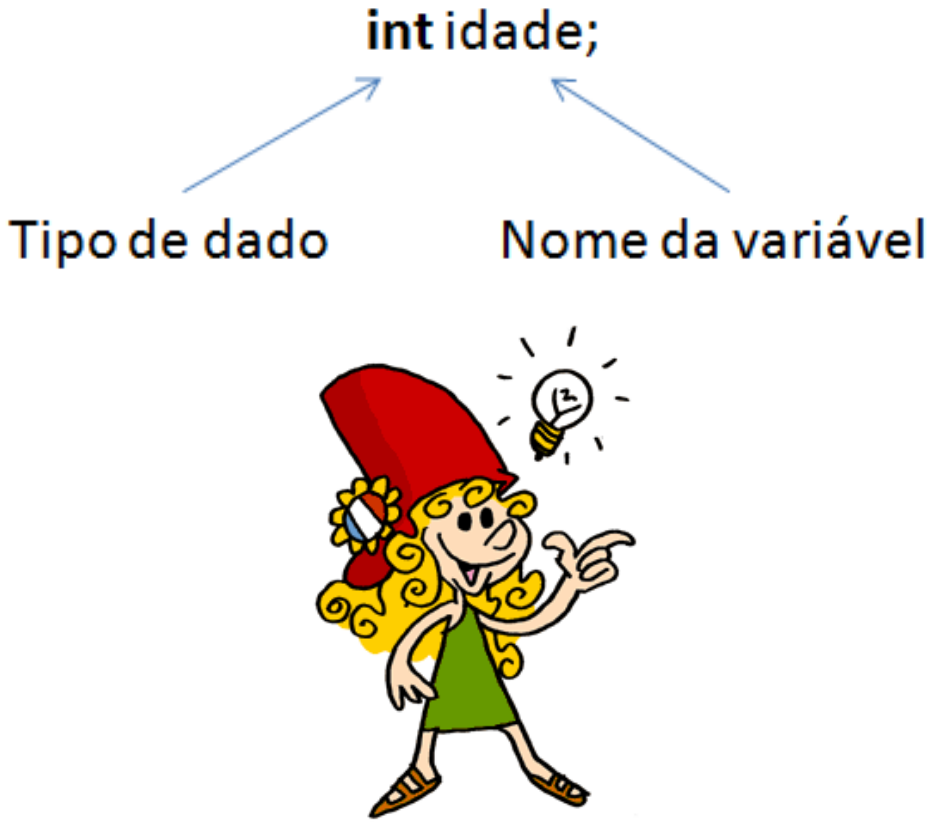
```c
/* Prog. C: Bom dia */
#include <stdio.h>

int main()
{
    printf("Bom Dia!!");
    return 0;
}
```

- *main()* é única, determina o inicio do programa.
- O comando *return* informa ao sistema operacional se o programa funcionou corretamente ou não.

Variáveis

- São locais onde armazenamos valores na memória.
- Toda variável é caracterizada por:
  - um nome: a identifica em um programa,
  - um tipo: determina o que pode ser armazenado naquela variável

Declarando uma variável I



Tipos de variáveis I

- Inteiras: utilizadas para armazenar valores inteiros

Tipo	**Tamanho**	**Valores possíveis**
(short) (signed) int	2 Bytes	-32.768 a 32.767
(short) unsigned int	2 Bytes	0 a 65.535
(signed) long int	4 Bytes	-2.147.483.648 a 2.147.483.647
unsigned long int	4 Bytes	0 a 4.294.967.295

Tipos de variáveis II

- Variáveis de tipo caracter
  - Utilizadas para armazenar letras e outro símbolos existentes em textos
  - São, na verdade, variáveis inteiras que armazenam um número associado ao símbolo. A principal tabela de símbolos utilizada pelo computadores é a tabela ASCII (*American Standard Code for Information Interchange*), mas existem outras (EBCDIC, Unicode, etc.)

Tipo	**Tamanho**	**Valores possíveis**
(signed) char	1 Byte	-128 a 127
unsigned char	1 Byte	0 a 256

Tipos de variáveis III

- Variáveis de tipo ponto flutuante
  - Armazenam valores reais, da seguinte forma

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

    Ex: $0{,}5 = (-1)^0 * 1 * 2^{-1}$.
  - Para o programador, funciona como se ele armazenasse números na forma decimal.
  - Possui problema de precisão (arredondamento).

Tipo	**Tamanho**	**Valores possíveis**
float	4 Bytes	$\pm 3,4E-38$ a $\pm 3,4E+38$
long float	8 Bytes	$\pm 1,7E-308$ a $\pm 1,7E+308$
double	8 Bytes	$\pm 1,7E-308$ a $\pm 1,7E+308$

Obtendo o tamanho de um tipo I

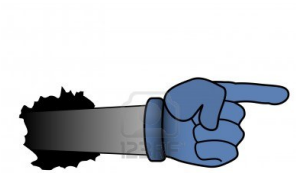
- O comando `sizeof`(tipo) retorna o tamanho, em *bytes*, de um determinado tipo. (Um *byte* corresponde a 8 *bits*).
- Ex. `printf("%d", sizeof(int));` ou `cout<<sizeof(int);`
- Escreve 4 na tela.

Obtendo o tamanho de um tipo II

Mostrar em *Bytes* o tamanho ocupado por variáveis do tipo inteiro, real e caracter.

```
#include <stdio.h>
int main(){
    printf("int      : %d bytes \n", sizeof(int));
    printf("long     : %d bytes \n", sizeof(long int));
    printf("float    : %d bytes \n", sizeof(float));
    printf("double   : %d bytes \n", sizeof(double));
    printf("char     : %d bytes \n", sizeof(char));
    return 0;
}
```

Regras para nomes de variáveis em C I



- Deve começar com uma letra (maiúscula ou minúscula) ou subscrito (_ *underscore*).
- **Nunca** pode começar com um número.
- Pode conter letras maiúsculas, minúsculas, número e subscrito
- Não pode-se utilizar `{ ( + - / \ ; . , ?` como parte do nome de uma variável.

Regras para nomes de variáveis em C II

- C/C++ são uma linguagem *case-sensitive*, ou seja, faz diferença entre nomes com letras maiúsculas e nomes com letras minúsculas: `Peso` e `peso` são diferentes.
- Costuma-se usar maiúsculas e minúsculas para separar palavras `PesoDoCarro`
- Identificadores devem ser únicos no mesmo escopo (não podem haver variáveis com mesmo identificador dentro do mesmo bloco).

Regras para nomes de variáveis em C III

- 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*

Regras para nomes de variáveis em C IV

- Quais dos nomes a seguir são nomes corretos de variáveis? Se não forem corretos, porque não são?

3ab	a3b	fim	int
`\meu`	`_A`	`n_a_o`	papel-branco
`a*`	`c++`	`*nova_variavel`	

Regras para nomes de variáveis em C V

- Exemplo 1:

```c
/* Exemplo de programa em C */
//  Arquivo de cabeçalho (header)
#include <stdio.h>
int main()
{
    int contador;  // declarações simples
    float PrecoDoQuilo;
    double TaxaDeCambio;
    char LetraDigitada;
    // Pode colocar mais de uma variável na
    // na mesma linha
    int IdadeManoel, IdadeJoao, IdadeMaria;
    double  TaxaDoDolar,     // Também pode trocar
            TaxaDoMarco,     // de linha no meio
            TaxaDoPeso,
            TaxaDoFranco;
.......
}
```

Regras para nomes de variáveis em C VI

- Exemplo 2: inicialização de variáveis

```
// Exemplo de programa em C
//  Arquivo de cabeçalho (header)
#include <stdio.h>
int main()
{
    // declara e inicializa com Zero
    int NroDeHoras = 0;
    // declara e inicializa com 10.53
    float PrecoDoQuilo = 10.53;
    double  TaxaDoDolar = 1.8,
            TaxaDoMarco = 1.956,
            TaxaDoPeso = 1.75,
            TaxaDoFranco = 0.2;


.......
    return 0;
}
```

Constantes I

- 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 seqüência de caracteres.
- Exemplos de constantes: 85, 0.10, 'c', "Meu primeiro programa"

Exemplo de declaração de constantes I

```
// Não se coloca ponto-e-vírgula após o valor
#define LARGURA_MAXIMA 50
#define NRO_DE_DIAS_DA_SEMANA 7
#define NRO_DE_HORAS_DO_DIA 24
#define VALOR_DE_PI  3.1415

int main ()
{
    int TotalDeHoras;
    const int r = 100;

    TotalDeHoras = 10 * NRO_DE_DIAS_DA_SEMANA *
                   NRO_DE_HORAS_DO_DIA;
.......
    return 0;
}
```

Escrevendo o conteúdo de uma variável na tela em C I

- Podemos imprimir, além de texto puro, o conteúdo de uma variável utilizando o comando *printf*.
- Utilizamos um símbolo no texto para representar que aquele trecho deve ser substituído por uma variável e, no final, passamos uma lista de variáveis ou constantes, separadas por vírgula.

Escrevendo o conteúdo de uma variável na tela em C II

- Ex.
  `printf ("A variável %s contém o valor %d", "a", a);`
- Imprime: A variável *a* contém o valor 10
- Nesse caso, `%s` deve ser substituído por uma variável ou constante do tipo *string* enquanto `%d` deve ser substituído por uma variável do tipo inteiro.

Formatos inteiro I

- %d: escreve um número inteiro na tela sem formatação

  Ex: `printf("%d", 10);`

  Imprime 10

Formatos inteiro II

- %< `número` >d: escreve um inteiro na tela, preenchendo com espaços a esquerda para que ele ocupe pelo menos < `número` > casas na tela.

  Ex: `printf("%4d", 10);`
  imprime ␣␣10

- %0< `número` >d: escreve um inteiro na tela, preenchendo com zeros a esquerda para que ele ocupe pelo menos < `número` > casas na tela.

  Ex: `printf("%04d", 10);`
  imprime 0010

Formatos inteiro III

- A letra *d* pode ser substituída pelas letras *u* e *l*, ou as duas, quando desejamos escrever variáveis do tipo *unsigned* ou *long*, respectivamente.

  Ex: `printf("%d", 4000000000);`

  escreve -294967296 na tela, enquanto que

  `printf("%u", 4000000000);`

  escreve 4000000000

Formatos ponto flutuante I

- `%f`: escreve um ponto flutuante na tela, sem formatação

  Ex: `printf("%f", 10.0);`

  imprime 10.000000

Formatos ponto flutuante II

- %e: escreve um ponto flutuante na tela, em notação científica
  - Ex: `printf("%e", 10.02545);`
  - imprime $1{,}002545e+01$

Formatos ponto flutuante III

- %< tamanho >.< decimais >f: escreve um ponto flutuante na tela, com tamanho < tamanho > e < decimais > casas decimais.
- O ponto utilizados para separar a parte inteira da decimal, também conta no tamanho

  Ex: `printf("%6.2f", 10.0);`
  ␣10.00

Formatos ponto flutuante IV

- A letra *f* pode ser substituída pelas letras *lf*, para escrever um *double* ao invés de um *float*

  Ex: `printf(%6.2lf), 10.0);`

  imprime ␣10.00

Formato caracter

- %c: escreve uma letra.

  Ex. `printf("%c", 'A');`
  imprime a letra A

  Note que `printf("%c", 65);` também imprime a letra A.

Formato *string* I

- %s: escreve um *string*

  Ex. `printf ("%s", "Meu primeiro programa");`
  imprime Meu primeiro programa

A função *scanf* I

- realiza a leitura de um texto a partir do teclado
- parâmetros:
  - uma *string*, indicando os **tipos das variáveis** que serão lidas e o formato dessa leitura.
  - uma **lista de variáveis**
- aguarda que o usuário digite um valor e atribui o valor digitado à variável

A função *scanf* II

```c
#include <stdio.h>
int main(){
    int n;
    printf("Digite um número: ");
    scanf("%d", &n);
    printf("O valor digitado foi %d \n", n);
    return 0;
}
```

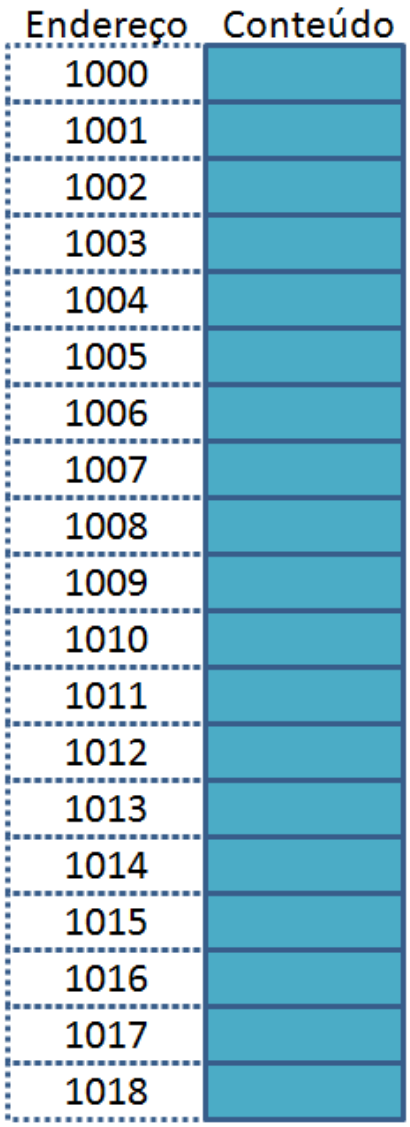
A função *scanf* III

O programa acima é composto de quatro passos:

- Cria uma variável n;
- Escreve na tela "Digite um número:"
- Lê o valor do número digitado
- Imprime o valor do número digitado

A função *scanf* IV

Memória



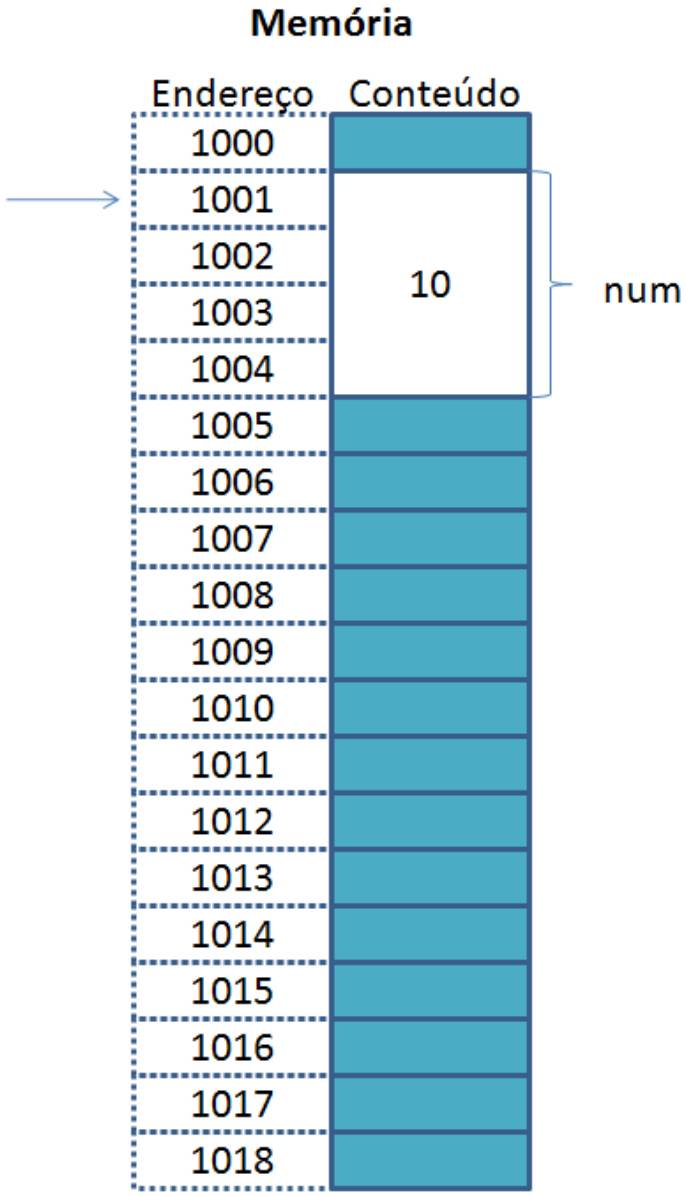
Código

```
int num = 10;  // 4 Bytes
char opcao = 's';  // 1 Byte
float nota;  // 4 Bytes
printf("Inserir nota");
scanf("%f", &nota);
```

Tela

Ação

A função *scanf* V



Código

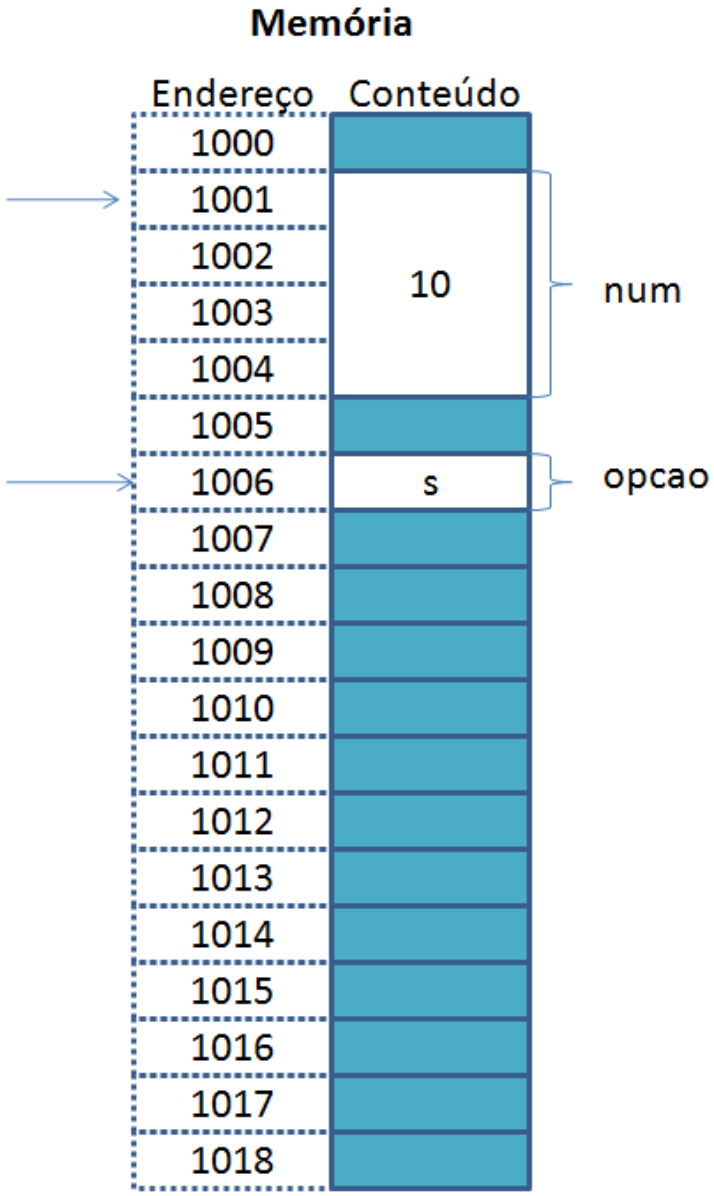
```
int num = 10;  // 4 Bytes
char opcao = 's';  // 1 Byte
float nota;  // 4 Bytes
printf("Inserir nota");
scanf("%f", &nota);
```

Tela

Ação

Declara e separa memória para a variável ***num*** do tipo inteiro
Atribui o valor 10

A função *scanf* VI



Código

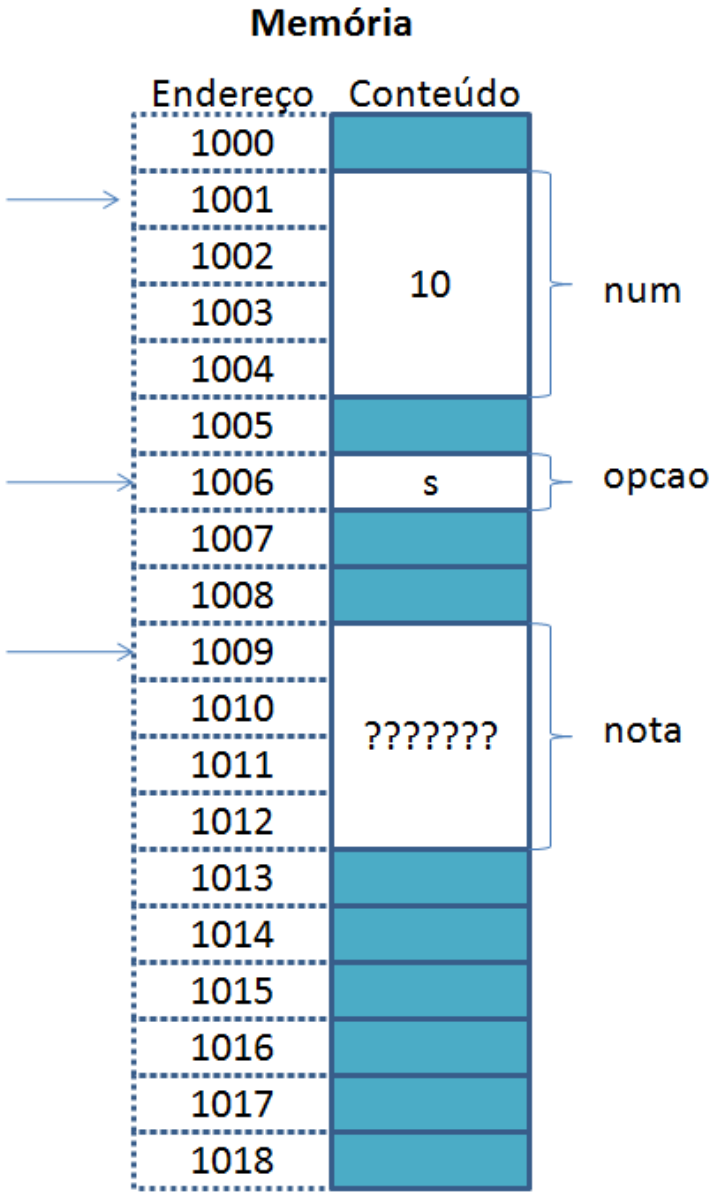
```
int num = 10;  // 4 Bytes
char opcao = 's';  // 1 Byte
float nota;  // 4 Bytes
printf("Inserir nota");
scanf("%f", &nota);
```

Tela

Ação

Declara e separa memória para a variável ***opcao*** do tipo caracter
Atribui o valor 's'

A função *scanf* VII



Código

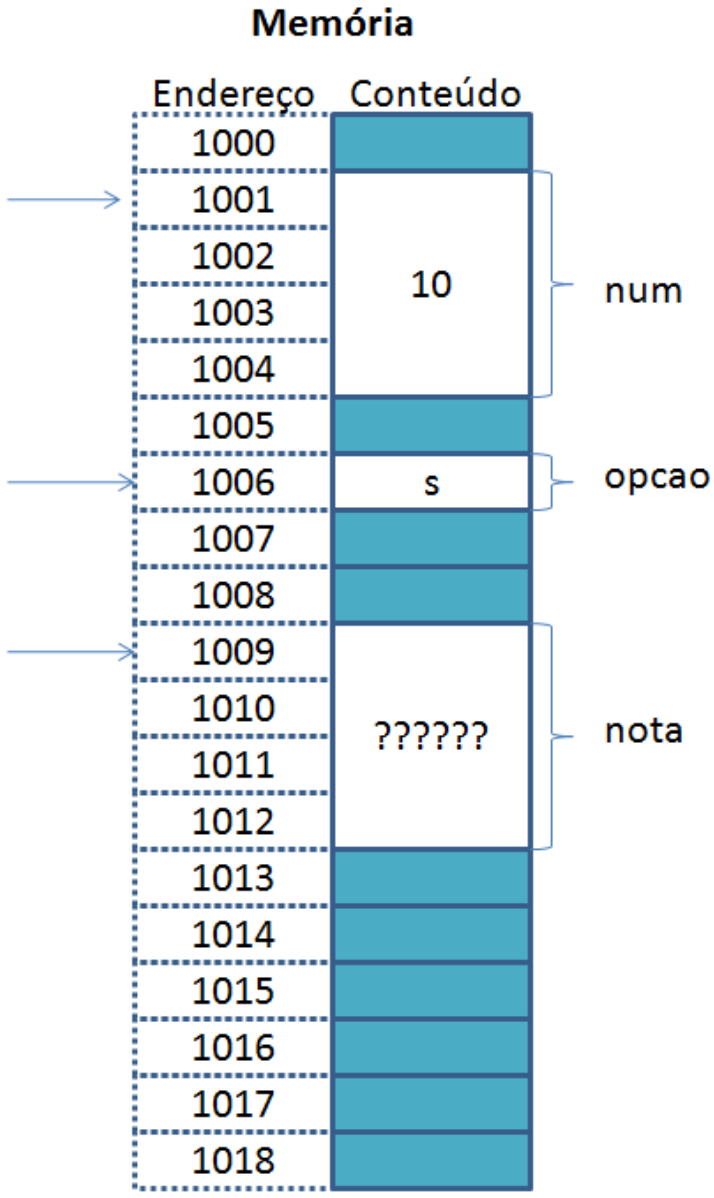
```
int num = 10;  // 4 Bytes
char opcao = 's';  // 1 Byte
float nota;  // 4 Bytes
printf("Inserir nota");
scanf("%f", &nota);
```

Tela

Ação

Declara e separa memória para a variável ***nota*** do tipo real de precisão simples

A função *scanf* VIII



Código

```
int num = 10;  // 4 Bytes
char opcao = 's';  // 1 Byte
float nota;  // 4 Bytes
printf("Inserir nota");
scanf("%f", &nota);
```

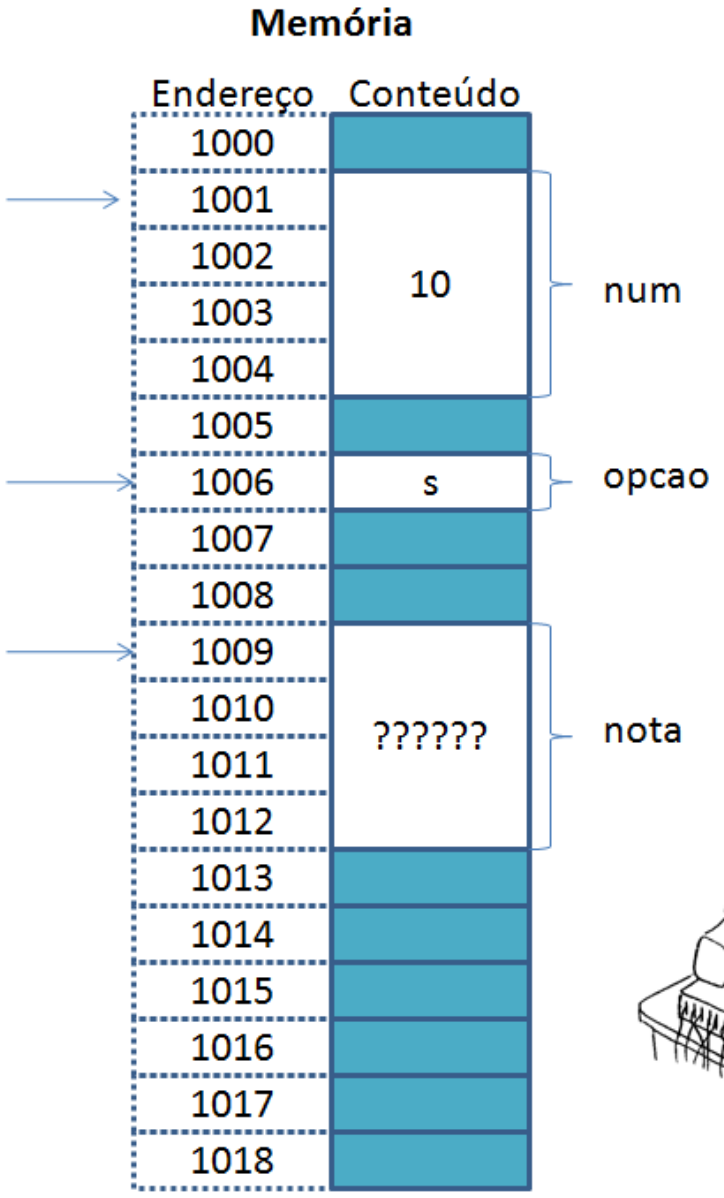
Tela

Inserir nota

Ação

Mostra na tela a mensagem : "Inserir nota"

A função *scanf* IX



Código

```
int num = 10;  // 4 Bytes
char opcao = 's';  // 1 Byte
float nota;  // 4 Bytes
printf("Inserir nota");
scanf("%f", &nota);
```

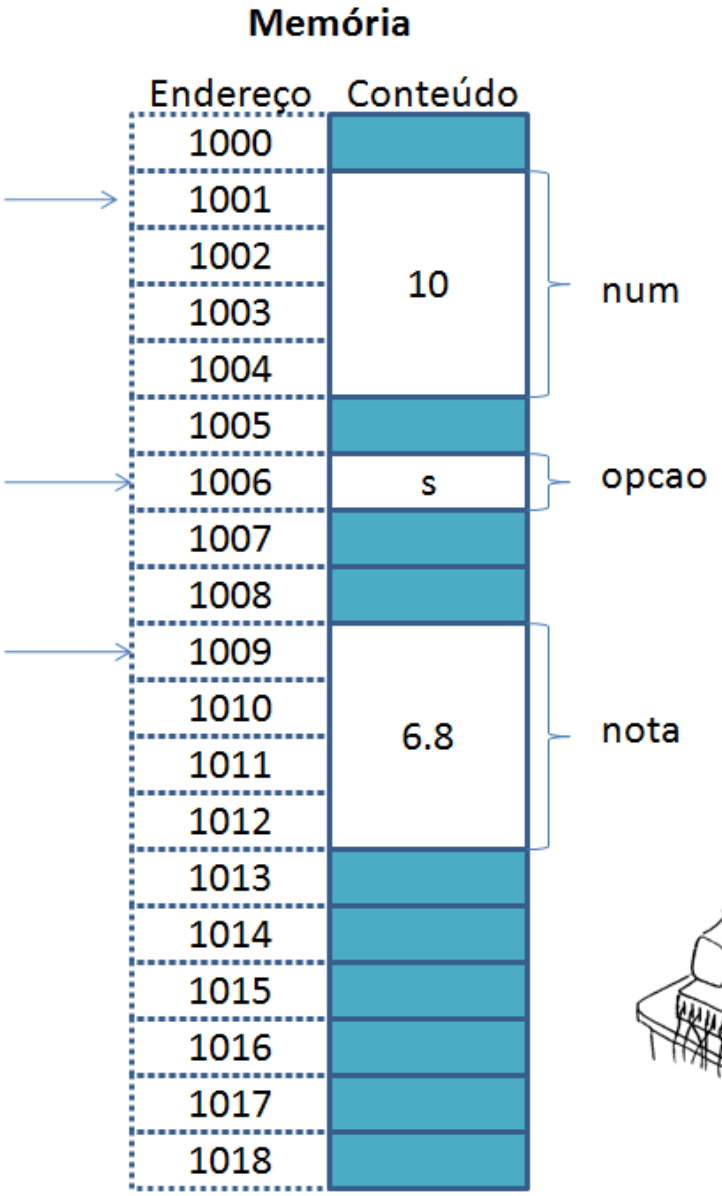
Tela

Inserir nota _

Ação

Operação de leitura, um valor número de tipo real será inserido no endereço de memória **1009**

A função *scanf* X



Código

```
int num = 10;  // 4 Bytes
char opcao = 's';  // 1 Byte
float nota;  // 4 Bytes
printf("Inserir nota");
scanf("%f", &nota);
```

Tela

Inserir nota 6.8

Ação

Usuário digita um número

A função *scanf* XI

Leitura de várias variáveis

```
#include <stdio.h>
int main(){
    int m, n, o;
    printf("Digite três números: ");
    scanf("%d %d %d",&m, &n, &o);
    printf("O valores digitados foram %d %d %d\n",
           m, n, o);
    return 0;
}
```

O endereço de uma variável I

- Toda variável tem um **endereço de memória** associada a ela.
- Esse endereço é o **local** onde essa variável é armanenada no sistema (como se fosse o endereço de uma casa, o local onde as pessoas são "armazenadas").

Endereço	Conteúdo	Variável
⋮	⋮	⋮
1001	???	
1002	2450	i
1003	???	
1004	225.345	f
1005	11331	j

O endereço de uma variável II

- Normalmente, o endereço das variáveis não são conhecidos quando o programa é escrito.
- O endereço 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.

O operador “address-of” & de C I

- O operador & retorna o endereço de uma determinada variável

```
printf("%x", &valor);
```

- Imprime o endereço da variável `valor`.

O operador "address-of" & de C II

- É necessário usar o operador & no comando `scanf`, pois esse operador indica que o valor digitado deve ser colocado no endereço referente a uma variável.
- Esquecer de colocar o & comercial é um erro muito comum que pode ocasionar erros de execução.

O operador “address-of” & de C III

O programa abaixo imprime o valor e o endereço da variável:

```
#include <stdio.h>
int main(){
    int n = 8;
    printf("valor %d, endereço %x \n", n, &n);
    return 0;
}
```

Formatos de leitura de variável I

Os formatos de leitura são muito semelhantes aos formatos de escrita utilizados pelo `printf`. A tabela a seguir mostra alguns formatos possíveis de leitura.

Código	Função
%*c*	Lê um único caracter
%*s*	Lê uma série de caracteres (*string*)

Formatos de leitura de variável II

Código	Função
$\%d$	Lê um número inteiro
$\%u$	Lê um número inteiro positivo
$\%l$	Lê um inteiro longo
$\%f$	Lê um número real de precisão simples
$\%lf$	Lê um número real de precisão dupla

Formatos de leitura de variável III

Digite um caracter na tela

```
#include <stdio.h>
int main(){
    char car;
    printf("Digite um caracter: ");
    scanf("%c", &car);
    printf("O caracter digitado foi: %c \n", car);
    return 0;
}
```

Formatos de leitura de variável IV

Crie um programa que permita inserir o nome e a idade de um aluno

```
#include<stdio.h>
int main(){
    char nome[50];
    int idade;
    printf("\n Inserir nome: ");
    scanf("%s", nome);
    printf("\n Inserir idade: ");
    scanf("%d", &idade);
    printf("\n %s tem %d anos ", nome, idade);
    return 0;
}
```

Formatos de leitura de variável V

Crie um programa que permita inserir o nome completo (nome e sobrenome) de um aluno e a idade

```
#include<stdio.h>
int main(){
    char nome[50];
    int idade;
    printf("\n Inserir nome completo: ");
    scanf("%s", nome);
    printf("\n Inserir idade: ");
    scanf("%d", &idade);
    printf("\n %s tem %d anos ", nome, idade);
    return 0;
}
```

O programa funciona quando é digitado um espaço em branco?

Formatos de leitura de variável VI

A função `scanf` não aceita espaços em branco. A função `fgets` aceita todo tipo de caracter, incluindo o espaço em branco.
Sintaxe:

```
char * fgets ( char *str, int num, FILE* stream );
```

```
#include<stdio.h>
int main(){
    char nome[50];
    int idade;
    printf("\n Inserir nome completo: ");
    fgets(nome, 50, stdin);
    printf("\n Inserir idade: ");
    scanf("%d", &idade);
    printf("\n %s tem %d anos ", %s, %d);
    return 0;
}
```

Formatos de leitura de variável VII

- O comando `fgets` também permite inserir espaços em branco
  Sintaxe: `char* gets ( char *str );`
- Não é recomendável o uso da função `gets`
- Não existe forma de controlar a quantidade de caracteres a serem inseridos
- Utilizar uma função mais segura como `fgets`

Atribuição I

Atribuir um valor de uma expressão a uma variável **significa calcular o valor** daquela expressão e **copiar** aquele valor para uma **determinada variável**.
O operador de atribuição é o sinal de igual (=)

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

=

A direita, deve haver uma expressão cujo valor será calculado e armazenado na variável.

Expressão I

- Uma expressão é um conjunto de operações aritméticas, lógicas ou relacionais utilizados para fazer "cálculos" sobre os valores das variáveis.
  Ex: `a + b`
  Calcula a soma de *a* e *b*

Expressão II

- 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 também é uma expressão
  Ex: `a = b;`
- `<expressão> + <expressão>`: calcula a soma de duas expressões.
  Ex. `a = a + b;`

Operadores I

- Aritméticos: `+ - * /` e o sinal negativo: `-` ;
- Relacionais:

Operador	**Significado**	**Operador**	**Significado**
&&	e (*and*)	||	ou (*or*)
!	não (*not*)	<	menor
<=	menor igual	>	maior
>=	maior igual	==	igual
! =	diferente		

- Operadores de Ponteiros: * (acessa o conteúdo do ponteiro), & (obtém o endereço de uma variável);

Operadores II

- Precedência é a ordem na qual os operadores serão calculados quando o programa for executado. Em C, os operadores são calculados na seguinte ordem:
  - * e /, na ordem em que aparecem na expressão.
  - %
  - + e −, na ordem em que aparecem na expressão.

Operadores III

- Incrementais: ++ (incrementa) ; -- (decrementa);
  - ++a: incrementa primeiro a e depois a usa (pré-fixado)
  - a++: primeiro usa a variável e depois a incrementa (pós-fixado)
- Atribuição, combinação dos outros operadores, são eles: $=$, $+=$, $-=$, $*=$, $/=$, %=

Incremento(++) e Decremento(- -) I

- Operadores de incremento e decremento tem duas funções: servem como uma expressão e incrementam ou decrementam o valor da variável ao qual estão associados em uma unidade. Ex: c++ incrementa o valor da variável *c* em uma unidade
- Dependendo da posição do operador de incremento e decremento, uma função é executada antes da outra.

Incremento(++) e Decremento(- -) II

- operador a direita da variável: Primeiro a variável é incrementada, depois a expressão retorna o valor da expressão. Ex:

```
#include <stdio.h>
int main () {
    int a = 10;
    printf ("%d", ++a);
    return 0;
}
```

Qual é valor de a?

Incremento(++) e Decremento(- -) III

- operador a direita da variável: Primeiro a variável é incrementada, depois a expressão retorna o valor da expressão. Ex:

```
#include <stdio.h>
int main () {
    int a = 10;
    printf ("%d", ++a);
    return 0;
}
```

Imprime 11

Incremento(++) e Decremento(- -) IV

- operador a direita da variável: Primeiro a expressão retorna o valor da variável, e depois a variável é incrementada. Ex:

```
#include <stdio.h>
int main (void) {
    int a = 10;
    printf ("%d", a++);
    return 0;
}
```

Incremento(++) e Decremento(- -) V

- operador a direita da variável: Primeiro a expressão retorna o valor da variável, e depois a variável é incrementada. Ex:

```
#include <stdio.h>
int main (void) {
    int a = 10;
    printf ("%d", a++);
    return 0;
}
```

Imprime 10

Incremento(++) e Decremento(- -) VI

- Em uma expressão, os operadores de incremento e decremento são sempre calculados primeiro (tem maior precedência)

```
#include <stdio.h>
    int main (void) {
    int a = 10;
    printf ("%d", a * ++a);
    return 0;
}
```

Qual é o valor de *a*?

Incremento(++) e Decremento(- -) VII

- Em uma expressão, os operadores de incremento e decremento são sempre calculados primeiro (tem maior precedência)

```
#include <stdio.h>
    int main (void) {
    int a = 10;
    printf ("%d", a * ++a);
    return 0;
}
```

Imprime 121

Incremento(++) e Decremento(- -) VIII

Quais são os valores impressos?

```
#include <stdio.h>
int main () {
   int a = 10;
   printf("%d\n", a++);
   printf("%d\n", a);
   return 0;
}
```

Atribuições simplificadas I

Uma expressão da forma

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

onde ocorre uma atribuição a uma das variáveis da expressão pode ser simplificada como

```
a += b;
```

Atribuições simplificadas II

Comando	Exemplo	Corresponde a:
$+=$	$a+=b$	$a=a+b;$
$-=$	$a-=b$	$a=a-b;$
$*=$	$a*=b$	$a=a*b;$
$\%=$	$a\%=b$	$a=a\%b;$

Atribuições simplificadas III

Qual é o valor mostrado na tela?

```
/*Operadores*/
int main(){
    int i = 10, j = 20;
    i = i + 1;
    i++;
    j -= 5;
    prinft("i + j = %d", i+j);
    return 0;
}
```

FIM