Linguagem C
Variáveis, Entrada e Saída

Profª. Rose Yuri Shimizu

Roteiro

Variáveis - armazenar valores

- $x + 4x.2 - (3/x)$
- O que é x?
  - Álgebra: variável que representa um número
  - Programação/Computação:
    - Variável que representa uma área de memória
    - Área de memória que armazena sequência de bits que representa um dado (número, letra, som, pixel, ...)

Variáveis - armazenar valores

- Referência a uma **informação** que é usada dentro do programa
- Representada por um símbolo/identificador
- Armazena números, letras

```
1 //ordem faz diferença!
2 c = 'A';    //c vale A
3 d = 'B';    //d vale B
4 d = c;      //d vale o que c vale
5 num = 123; //num vale 123
```

- As variáveis precisam ser **declaradas** e **inicializadas**
- **ANTES DE USAR, TEM QUE DECLARAR**

Variáveis - declaração

- Indicar o tipo de dados armazenado e o nome (identificador)

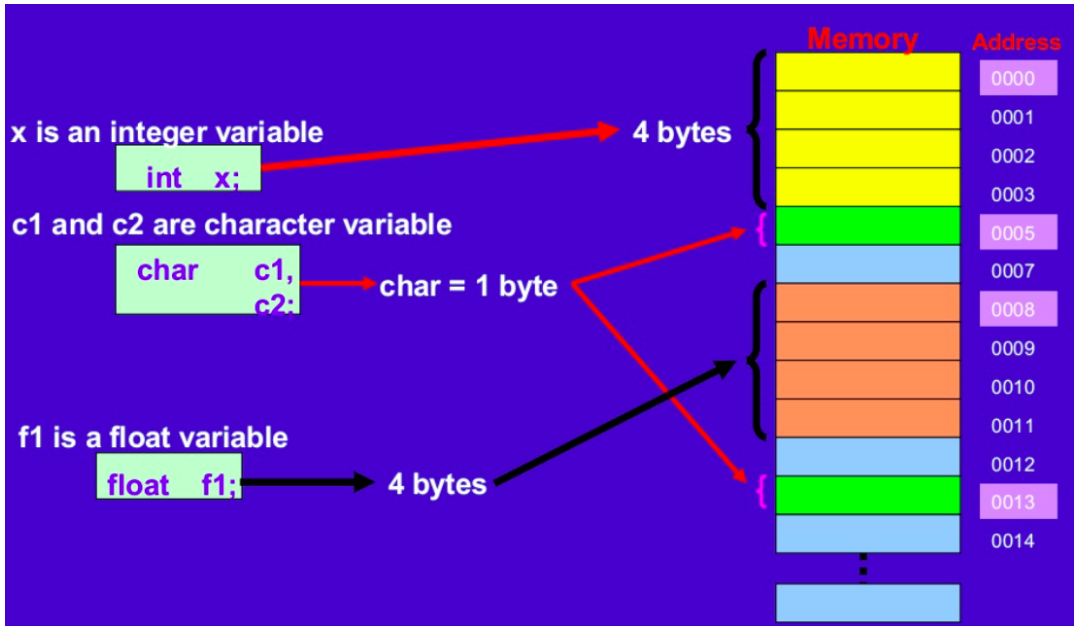
```c
//"Bibiblioteca padrão de entrada/saída" do C (libc) - C11/C17/C23
#include <stdio.h>

//main: função principal é a primeira instrução executada
//      início da execução
int main()
{
    int x; //declarando a variável x do tipo inteiro

    char letra; //declarando a variável letra do tipo caractere

    int a, b, c; //várias declarações do tipo inteiro


    //padrao C: return 0 sucesso e não-zero erro
    return 0;
}
```

Variáveis - tipos de dados

- **int**: armazena números inteiros
- **char**:
  - Armazena caracteres
  - Símbolo (uma letra do alfabeto, um dígito, um símbolo de pontuação, etc)
  - Cada caractere (ocupa 1 byte) é associado um valor (256 possibilidades - 2^8)
  - Representado entre apóstrofes (')
    - ★ 'C', 'a', '5', ''$'
    - ★ '5' é um caractere, e não um inteiro 5
- **float**(precisão simples) e **double**(precisão dupla):
  - Armazena números reais (números com o ponto decimal)

Variáveis - tipos de dados

- Indica quantos bytes será alocado/reservado na memória
- Cada tipo ocupa um número específico de bytes



Variáveis - representação de números

- Representar números na base 2 (sistema binário)
- $101b = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 5_d$
- A quantidade de bits(n) indica quantos números podem ser representados: 2^n números distintos:

Decimal	Binário (2 bits $\rightarrow 2^2 = 4 \rightarrow$ 0-3)	3 bits $\rightarrow 2^3 = 8 \rightarrow$ 0-7
0	00	000
1	01	001
2	10	010
3	11	011
4	-	100
5	-	101
6	-	110
7	-	111

- 1 BYTE: grupo de 8 bits
- Quanto mais números queremos representar, mais bits são necessário.

Variáveis - representação de caracteres

- Todos os dados são representados por números
- Cada caractere é representado por 8 bits $\rightarrow$ 1 byte
- Cada caractere $\rightarrow$ número;
- Segue a tabela ASCII

Binário	Caracter	Binário	Caracter	Binário	Caracter
0010 0001	!	0100 0001	A	0110 0001	a
0010 0010	"	0100 0010	B	0110 0010	b
0010 0011	#	0100 0011	C	0110 0011	c
0010 0100	$	0100 0100	D	0110 0100	d
0010 0101	%	0100 0101	E	0110 0101	e
0010 0110	&	0100 0110	F	0110 0110	f
0010 0111	'	0100 0111	G	0110 0111	g
0010 1000	(	0100 1000	H	0110 1000	h
0010 1001	)	0100 1001	I	0110 1001	i

Variáveis - tipos de dados - intervalos

Tipo	Tamanho	Intervalo ≈
char	8 bits	-128 → 127
short	16 bits	-32 mil → 32 mil
unsigned short	16	0 → 65 mil
int	32 bits	-2 bilhões → 2 bilhões
unsigned int	32	0 → 4 bilhões
long	64 bits	-9 quintilhões → 9 quintilhões
long long	64 bits	-9 quintilhões → 9 quintilhões
unsigned long	64	0 → 18 quintilhões
float	32 bits	$-3.4x10^{38} \rightarrow 3.4x10^{38}$
double	64 bits	$-1.7x10^{308} \rightarrow 1.7x10^{308}$

- float: 6 dígitos de precisão decimal (restante é aproximação)
- double: 15 dígitos de precisão decimal (restante é aproximação)
- Exemplo:
  - float 1**123456**123456123456 = 1**123456**104810938368
  - double 1**1234561234561234**56 = 1**1234561234561234**92

Variáveis - tipos de dados - memória

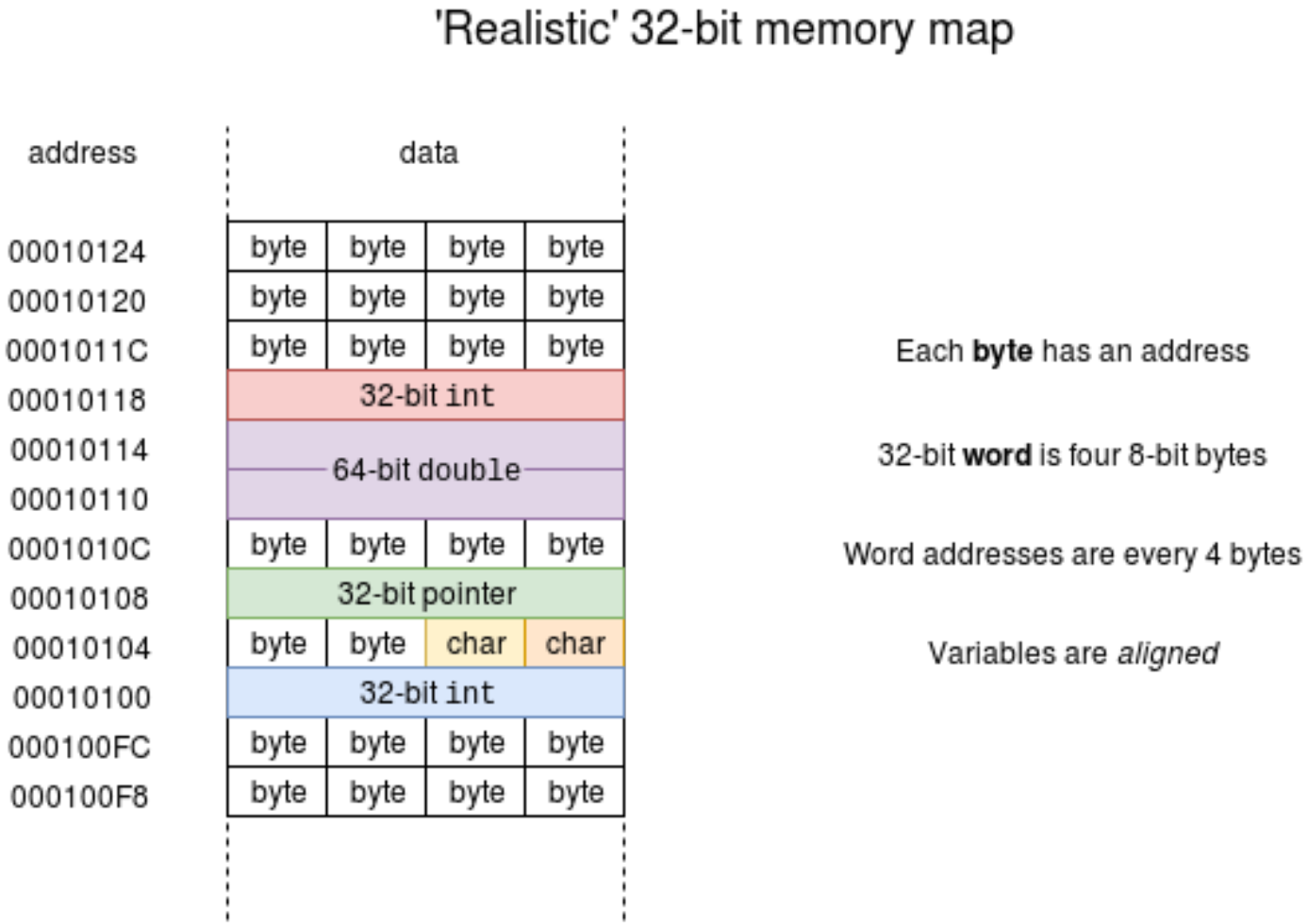


Figura: https://xerxes.cs.manchester.ac.uk/comp251/kb/images/variables.png

Variáveis - identificadores/nomes

- DEVE COMEÇAR com uma letra ou com '_';
- Seguido de qualquer sequência de letras, digitos, e '_';
  - hora_inicio, tempo, var1 são nomes de variáveis válidos;
  - 3horas, total$ e azul-claro são nomes inválidos.
- Maiúsculas $\neq$ Minúsculas $\rightarrow$ *case sensitive*;
- Não são permitidos nomes ou palavras reservadas da linguagem (char, const, continue, default, do, double, else, etc.)

Variáveis - inicialização por atribuição

- Valores às variáveis usando o **operador de atribuição (=)**

```c
int main()
{
    //Declaracao de variaveis
    int pera;
    char qualidade;
    float peso;

    //Inicializacao de variaveis por atribuicao
    pera = 3;
    qualidade = 'A';
    peso = 0.653;

    //Inicializacao na declaracao
    int maca = 1;
}
```

Variáveis - compatibilidade de dados

Se tentarmos colocar um valor diferente do tipo esperado da variável? Alguns casos:

- **Declaramos um int e colocamos um char**
  - Os literais de caracteres são, nativamente, do tipo int;

```c
/* ASCII (man ascii) */
int num = 'A';
printf("%d\n", num); //A??
```

- **Declaramos um int e colocamos um float**

```c
int num = 77.3;
printf("%d\n", num); //Saída 77: perde-se a parte decimal
```

- **Overflow - declaramos um short e colocamos um valor maior que o máximo**
  - Valor incorreto
  - Fiquem atentos a capacidade de cada tipo, senão pode computar incorretamente mesmo com a lógica correta do programa.

```c
short num = 33000;
printf("%d\n", num); //Saída -32536
```

Variáveis - resumo

- Linguagem C é fortemente tipada
- **ANTES DE USAR, TEM QUE DECLARAR**
- **TODA INSTRUÇÃO TERMINA COM ';'**
- **Variáveis**, associadas:
  - **Um tipo**: quantos bytes ocupa, e como ela deve ser interpretada
  - **Um nome**: um identificador
  - **Um endereço**: o endereço do byte menos significativo do local da memória associado a variável
  - **Um valor**: o conteúdo da variável

Variáveis - quais são os erros?

```c
#include <stdio.h>
int main()
{
    //Declaracao de variaveis
    int beija-flor;
    int pera;
    float 9p;
    float peso
    char A;

    //Inicializacao de variaveis por atribuicao
    pera = 3;
    A = 'A';
    A = "B";
    peso = 0,653;

    //Inicializacao na declaracao
    int maca = 1;

    return 0;
}
```

Variáveis - cópia de variáveis

```c
#include <stdio.h>
int main()
{
    //Declaracao de variaveis
    int A;
    float B;
    char C;

    //Quais os conteúdos de A, B e C?

    //Inicializacao de variaveis por atribuicao
    A = 68;
    B = 9.9;
    C = 'p';

    //Quais os conteúdos de A, B e C?

    C = A;
    A = B;

    //Quais os conteúdos de A, B e C?

    return 0;
}
```

Roteiro

Constantes x Variáveis

- São números ou caracteres cujos valores não mudam
- Assim como variáveis, constantes também tem tipos (int, char, etc.)
- Podem ser utilizadas diretamente:
  - 2 e 9 são constantes e não precisam ser declaradas

```
1 int a; //declaração da variável
2 a = 2+9;
```

- Podem ser declaradas através do modificador de acesso `const`:

```
1 const float num_pi = 3.14159;
2 int a, b;
3 a = 10 + num_pi;
4 b = num_pi + a;
```

Constantes x Variáveis

- Diretiva #define <nome> <valor>

```c
#include <stdio.h>
/* Compilacao: eh feito um pre-processamento que substitui toda
     referencia do nome no codigo substituicao das macros pelos
     seus valores */

//declaracao da constante
#define ICMS 0.18

int main()
{
    float preco_produto, valor_icms;
    preco_produto = 150;
    valor_icms = preco_produto * ICMS;
    printf("Valor do icms %f \n", valor_icms);
    return 0;
}
```

Roteiro

Função **printf()** - saída

```c
#include <stdio.h>

int main()
{
    //Imprime mensagens na tela
    //Imprime tudo que esta entre aspas dupla
    //Toda instrução termina com ";"
    //\n nova linha e fim de linha

    printf("Alo mundo\n");

    printf("Alo");
    printf(" mundo\n");

    printf("Alo"
           " mundo\n");

    printf("Alo");
    printf(" ");
    printf("mundo\n");

    return 0;
}
```

Função **printf()** - imprimir variáveis

```c
#include <stdio.h>

int main()
{
    int X = 3;
    int Y = 2;

    //         |---------|
    printf("%d %d\n", X, Y);
    //              |--------|

    return 0;
}
```

- **Especificadores de formato**:
  - Tipo do dado que será substituído pelo valor da variável

Tipo	Especificador
int	%d
long	%ld
char	%c
float	%f
double	%lf

Função **printf()** - imprimir mensagens + variáveis

```c
#include <stdio.h>

int main()
{
    int X = 3;
    int Y = 2;

    //                |------------------|
    printf("X vale %d e Y vale %d\n", X, Y);
    //                                  |--------|

    //Saída: X vale 3 e Y vale 2

    return 0;
}
```

Tipo	Especificador
int	%d
long	%ld
char	%c
float	%f
double	%lf

```c
#include <stdio.h>

int main()
{
    int a = 10;
    float c = 3.12;
    double d = 555121.42;

    char letra;
    letra = 'e';

    printf("Inteiro %d \n",a);
    printf("Caractere %c \n",letra);
    printf("Float %f \n",c);
    printf("Double %lf \n",d);
    printf("Expressoes %c = %d + %d \n", letra, a, a);

    printf("Ola Mundo!\n");
    printf("%c %d\n", e, e);

    return 0;
}
```

- Salvar como "meuprograma.c"
- Compilar: gcc meuprograma.c -o meuprograma
- Executar: ./meuprograma

Função **printf()** - formatar a saída

- Imprime um número fixo de casas inteiras (int): %*w*d
- Imprime um número fixo de casas decimais (float, double): %.*w*f

```c
#include <stdio.h>
int main()
{
    int x = 1, y = 20;
    printf("%5d%5d\n", x, y);      //    1   20

    float z = 123.456789;
    double p = 123.456789;
    printf("%.1f %.3lf\n", z, p); //123.5 123.457

    return 0;
}
```

Função **printf()** - retorno

- Funções podem RETORNAR valores
- A função `printf` devolve quantos caracteres foram impressos
- Devolve para a função que a chamou/invocou

```c
#include <stdio.h>

int main()
{
    int i = printf("alo mundo\n");
    printf("%d\n", i); //saída: 10 (?)

    return 0;
}
```

Função **printf()** - variável x endereço

- Cada variável possui um endereço na memória
- Cada *byte* possui um endereço
- Endereço = *byte* menos significativo (início da alocação)

```
int x = 9;
```

$9_{10} = 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 1001_b$

```
int: ocupa 32 bits (4 bytes)
&x : 6024 (endereço hipotético)
```

Memória	00000000	00000000	00000000	00001001
Endereço	6027	6026	6025	6024

Função **printf()** - endereço **&**

```c
#include <stdio.h>

int main(){
    int i = 10;

    printf("%d\n", i);          //saída?
    printf("%p\n", &i);         //%p hexadecimal
                                //0x7ffebbe8f8bc

    printf("%ld\n",(long)&i); //(long) typecast : conversão de tipo
                                //140732051028156

    return 0;
}
```

Roteiro

Função **scanf()** - entrada

- Função "scanf" é a responsável por ler a entrada do teclado
  - Teclado = entrada padrão
- Ler do teclado e gravar o valor nas variáveis
  - Grava em um endereço
  - As variáveis são passadas por **referência**: pelo endereço da variável
  - O endereço é obtido pelo símbolo &
- O que será lido é definido através dos especificadores de formato:
  - %d int
  - %ld long int
  - %c char
  - %f float
  - %lf double (dupla precisão)

```
1 int i;
2 scanf("%d", &i);
```

Função **scanf()** - entrada

Exemplos de entradas

code:

```
scanf("%d",&x)
```

run:

=40 ↵

x

40

code:

```
scanf("%d%d",&x,&y);
```

run1:

=40 ↵

=80 ↵

run2:

=40 80 ↵

x

40

y

80

Função **scanf()** - entrada

```c
#include <stdio.h>

int main()
{
    //Declaracao de variaveis
    float y;
    int x;

    //Inicializacao de variaveis pela entrada padrao
    //Lê a entrada até consumir todos os tipos indicados na função
    //Caracteres separadores (‘ ’, ‘\n’, ‘\t’) serão ‘‘ignorados’’
    //          até a próxima entrada
    scanf("%f %d", &y, &x); //formas de entradas válidas:
                            // 23 24 <enter> : tudo na mesma linha
                            // 23 <enter>    : linhas distintas
                            // 24 <enter>

    //Saida
    printf("Voce digitou %f %d\n", y, x);
    return 0;
}
```

Função **scanf()** - entrada

```c
#include <stdio.h>

int main()
{
    int a;

    scanf("Num %d", &a); //entrada exata: Num 3 <enter>

    scanf("%d\n",&a);
    scanf("%d ",&a);
          //^ não deixe espaço ou \n entre o última entrada e o "

    return 0;
}
```

Função **scanf()** - entrada

```c
#include <stdio.h>
int main() {
    char b, c, d;

    //leituras seguidas de caracteres
    scanf("%c", &b);
    scanf("%c", &c);

    scanf(" %c", &d);
        //^ deixe espaço entre o última entrada e o "

    printf("%c\n", b);
    printf("%c\n", c);
    printf("%c\n", d);

    return 0;
}
```

- Entradas:
  - x y z
- Saída?

Função **scanf()** - entrada

```c
#include <stdio.h>
int main() {
    char b, c, d;

    //leituras seguidas de caracteres
    scanf("%c", &b);
    scanf("%c", &c);

    scanf(" %c", &d);
        //^ deixe espaço entre o última entrada e o "

    printf("%c\n", b);
    printf("%c\n", c);
    printf("%c\n", d);

    return 0;
}
```

- Entradas:
  - x y z
- Saída?
  1. x

Função **scanf()** - entrada

```c
#include <stdio.h>
int main() {
    char b, c, d;

    //leituras seguidas de caracteres
    scanf("%c", &b);
    scanf("%c", &c);

    scanf(" %c", &d);
        //^ deixe espaço entre o última entrada e o "

    printf("%c\n", b);
    printf("%c\n", c);
    printf("%c\n", d);

    return 0;
}
```

- Entradas:
  - x y z
- Saída?
  1. x
  2. <espaço em branco>

Função **scanf()** - entrada

```c
#include <stdio.h>
int main() {
    char b, c, d;

    //leituras seguidas de caracteres
    scanf("%c", &b);
    scanf("%c", &c);

    scanf(" %c", &d);
        //^ deixe espaço entre o última entrada e o "

    printf("%c\n", b);
    printf("%c\n", c);
    printf("%c\n", d);

    return 0;
}
```

- Entradas:
  - x y z
- Saída?
  1. x
  2. <espaço em branco>
  3. y

Função **scanf()** - entrada

```c
#include <stdio.h>
int main() {
    char b, c, d;

    //leituras seguidas de caracteres
    scanf("%c", &b);
    scanf("%c", &c);

    scanf(" %c", &d);
        //^ deixe espaço entre o última entrada e o "

    printf("%c\n", b);
    printf("%c\n", c);
    printf("%c\n", d);

    return 0;
}
```

- Entradas:
  - x y z
- Saída?
  1. x
  2. <espaço em branco>
  3. y

- `scanf` lê todos os caracteres inclusive os separadores
  - espacos, tab, `\n`
- "Consumir" os separadores anterior: utilize um espaço

Função **printf()** e **scanf()**

printf

A função **printf()** só precisa do valor da variável para imprimi-la. Acessa diretamente o dado pelo nome da variável.

```
1 int x;
2 x = 10;
3 printf("%d", x);
```

scanf

A função **scanf()** precisa saber o endereço da variável para gravar dados dentro dela. Acessa o endereço através do operador &.

```
1 int x;
2 scanf("%d", &x);
```

Função **scanf()** - formatar a entrada

- Lendo um número fixo de casas inteiras: %*w*d %*w*f

```c
#include <stdio.h>
int main()
{
    int x;
    float y;

    scanf("%3d %2f", &x, &y);
    printf("%d %f\n", x, y);

    return 0;
}
```

- Execução 1: 345 78
  - scanf : irá ler os 3 digitos de 345 e os 2 digitos de 78
  - printf : 345 78.000000
- Execução 2: 2654 87
  - scanf : irá ler os 3 digitos 265 e gravar na variável x e o 4 será a próxima entrada e gravada em y
  - printf : 265 4.000000

Função **scanf()** - retorno

- A função `scanf` devolve quantos caracteres foram lidos
- Devolve para a função que a chamou/invocou

```c
#include <stdio.h>

int main() {
    int a;
    char b;
    float c;

    int i = scanf("%d %c %f", &a, &b, &c);

    printf("%d\n", i); //saída?

    return 0;
}
```

- Salvar como "meu_programa.c"
- Compilar: gcc meu_programa.c -o meu_programa
- Executar: ./meu_programa

Exercícios de fixação

1. Faça um algoritmo com duas variáveis inteiras, com os valores 12 e 33, imprima seus valores na tela.
2. Faça um algoritmo que imprima a mensagem "Programar é fácil!".
3. Faça um algoritmo com uma variável do tipo caractere para cada letra do seu nome e imprima a mensagem: "Meu nome é Fulano". Sendo que o seu nome deve ser impresso no lugar da palavra "Fulano" através do conteúdo das variáveis.
4. Faça um algoritmo com duas variáveis reais, leia da entrada padrão e imprima na tela os valores lidos.
5. Faça um algoritmo com duas variáveis do tipo caractere, leia da entrada padrão e imprima na tela os valores lidos.
6. Faça um algoritmo uma variável do tipo int, uma do tipo float, uma do tipo double, uma do tipo char. Inicialize por atribuição (utilizando =) e imprima na tela. Depois, inicialize novamente mas pela entrada padrão, e imprima novamente.

- Ambientes: `https://fga.rysh.com.br/apc/index.html?#ambientes`