Variáveis, Entrada e Saída
Linguagem C

Profª. Rose Yuri Shimizu

Roteiro

Linguagem de programação C

- Linguagem C nasceu na década de 70
- Inventado por Dennis Ritchie, reescreveu o sistema operacional UNIX
  - Originalmente UNIX foi implementado em assembly
- Derivado da linguagem B, criado por Ken Thompson
- Linguagem propósito geral
  - Implementa instrução da arquitetura (linguagem de máquina)
  - Desenvolvimento de programas diversos como processadores de texto, planilhas eletrônicas, sistemas operacionais, programas para a automação industrial, gerenciadores de bancos de dados, etc.

Linguagem de programação C

- Compilada
  - Possui conjunto limitado de palavras (*vocabulário*)
  - Associado a um conjunto de regras (*sintaxe*)
    - Palavras válidas: palavras reservadas, identificadores, operadores, etc - *análise léxica*
    - Combinações de palavras formam instruções válidas (comandos, expressões, funções, etc) - *análise sintática*
  - Traduzida para linguagem de máquina pelo compilador
- Imperativa: ordens através de instruções
- Estruturada
  - Sequencial: a tarefa é executada logo após a outra;
  - Condicional (decisão): a tarefa é executada após um teste lógico, e;
  - Iterativa (repetição): a partir do teste lógico, um trecho do código pode ser repetido finitas vezes.
- Procedural: blocos/trechos de códigos com função específicas, que podem ser reutilizados

Roteiro

Programas C - Estrutura básica

- **stdio.h**:
  - É um cabeçalho da biblioteca (coleção de funções/informações) padrão do C (libc) - C11/C17/C23
  - "Cabeçalho padrão de entrada/saída";
  - Exemplos: leitura do teclado e impressão na tela, definições de constantes, variáveis e tipos.
- Função **main**: principal - **início de execução do programa**
- **Programa sequencial**: as instruções são executadas sequencialmente
- **TODA INSTRUÇÃO TERMINA COM ";"**

```
1 #include <stdio.h>
2
3 int main() {
4     ...
5     return 0; //padrao C: return 0 sucesso e nao-zero
      erro.
6 }
```

Roteiro

Expressão algébrica

- $x + 4x.2 - (3/x)$
- O que é x?

Expressão algébrica

- $x + 4x.2 - (3/x)$
- O que é x?
  - Álgebra: variável que representa um número

Expressão algébrica

- $x + 4x.2 - (3/x)$
- 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

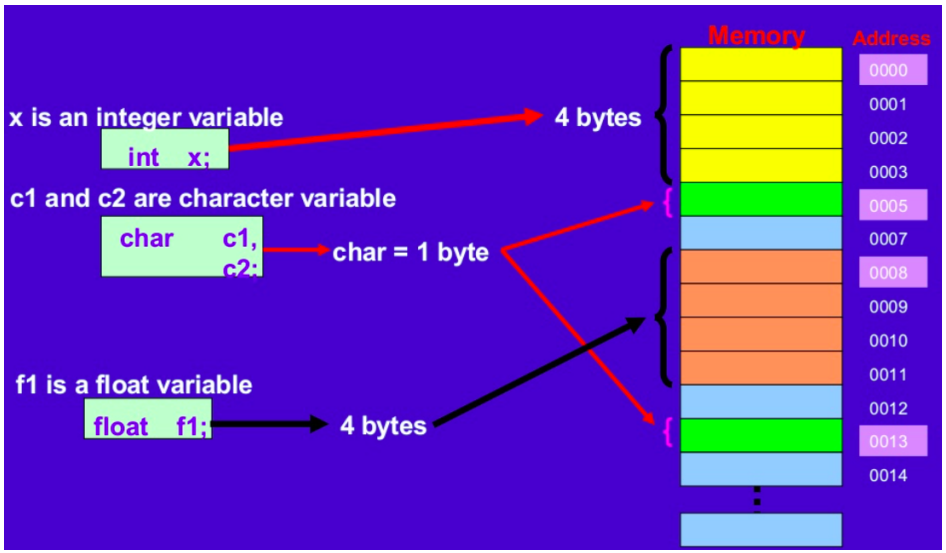
- Uma variável é uma **informação** que é usada dentro de um programa C
- É um símbolo representativo capaz de representar números, letras, palavras

```
1 c = 'A';
2 num = 123;
3 minhaVariavel = 5;
```

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

Declaração de variáveis

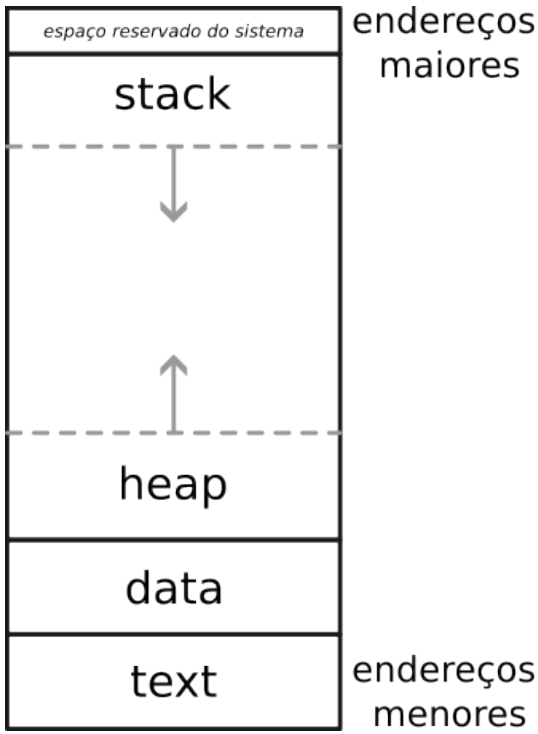
- Indicar o tipo de dados armazenado e o nome (identificador)
  - Sintaxe: **tipo *nome_variavel*** ;
  - **tipo**
    - ★ Indica quantos bytes será alocado/reservado na memória
    - ★ Cada tipo ocupa um número específico de bytes
    - ★ Cada tipo tem um formato específico (significado de cada conjunto de bits - ex.: sinal, decimal)



  - ***nome_variavel***
    - ★ Representa um endereço (localização) de memória para acessar um valor/conteúdo (pode mudar)

Declaração de variáveis - memória

- Programa em execução: processo
- Cada processo: possui uma porção da memória
- Cada porção: organizada por segmentos
- Layout geral dos segmentos:



- stack:
  - Variáveis locais, parâmetros de funções e endereços de retorno (instrução que chamou uma determinada função)
  - Tamanho: limitado pelo SO
  - Linux: 8192 kB (ulimit -s)
- heap: blocos de memória alocadas dinamicamente, a pedido do processo (gerenciado pelo sistema operacional)
- data: variáveis globais e estáticas
- text: código que está sendo executado

Declaração de variáveis - Exemplos

```c
#include <stdio.h>

int main()
{
    //comentário: duas barras juntas

    int x; //declarar

    char letra; //declarar

    int a, b, c; //várias declarações

    return 0;
}
```

Declaração de 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)
- Todas os dados são representados por números

Declaração de 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.

Declaração de variáveis - Representação de Caracteres

- 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

Declaração de variáveis - Tipos de dados - Intervalos

Tipo	Tamanho	Intervalo
char	8 bits	-128 - 127
short	16 bits	-32768 - 32767
unsigned short	16	0 to 65535
int	32 bits	-2147483648 - 2147483647
unsigned int	32	0 to 4294967295
long	64 bits	-9223372036854775808 - 9223372036854775807
long long	64 bits	-9223372036854775808 - 9223372036854775807
unsigned long long int	64	0 to 18446744073709551615
float	32 bits	-1.17549e-38 - 3.40282e+38
double	64 bits	-2.22507e-308 - 1.79769e+308

- float: 6 dígitos de precisão decimal
- double: 15 dígitos de precisão decimal

Declaração de variáveis - Tipos de dados - Memória

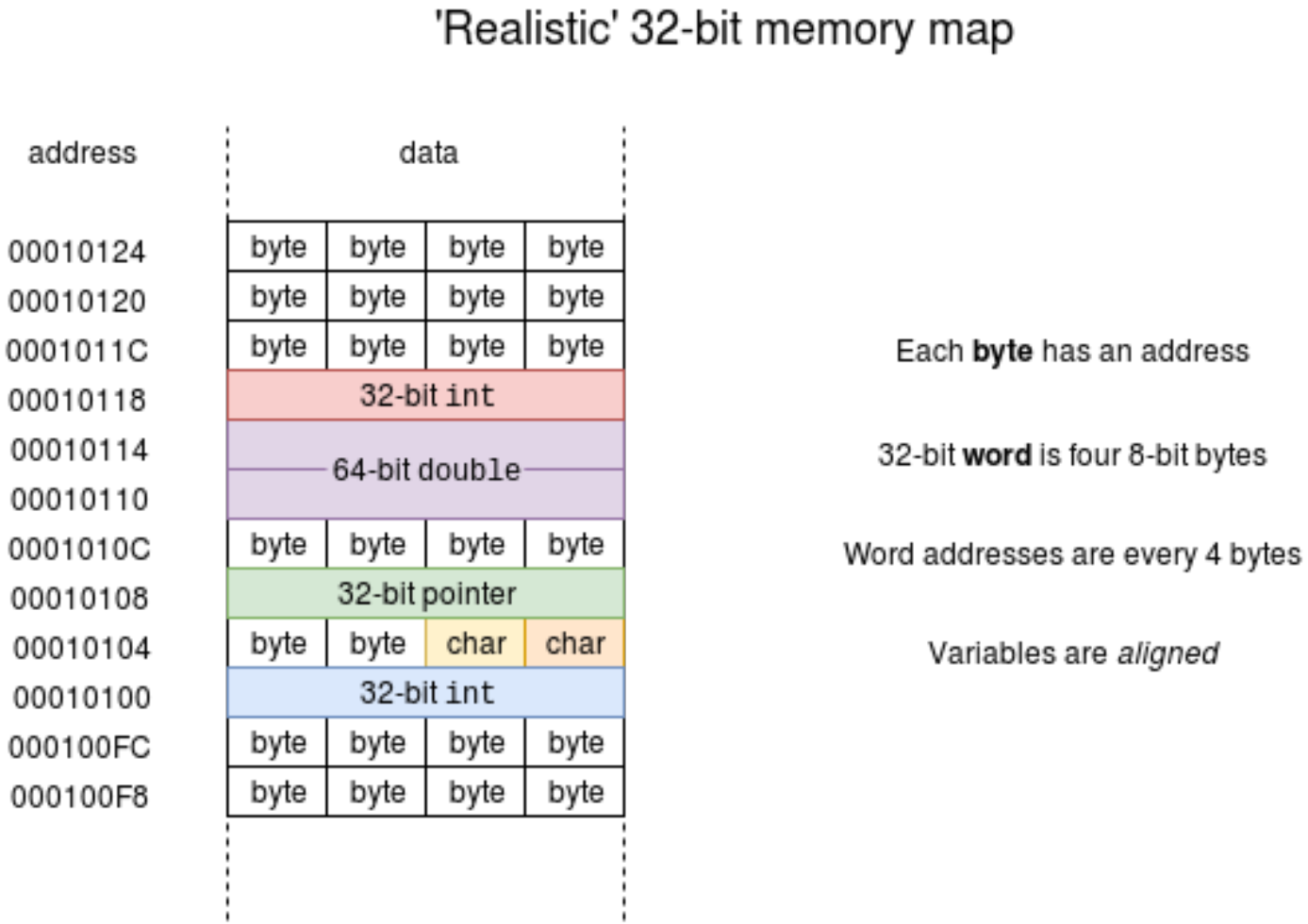


Figura: https://xerxes.cs.manchester.ac.uk/comp251/kb/images/variables.png

Declaração de 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.)

Inicialização de variáveis (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;
}
```

Inicialização de variáveis (atribuição) - 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**
  - 77.33 $\rightarrow$ 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.

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 - Atribuição 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
  - Obs.: 'a' é a variável, então precisa ser declarada

```
1 int a;
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()**

- Pode ser utilizado para **imprimir na tela mensagens**
- **Imprimir mensagens**:

```
1 //imprime tudo que esta entre aspas dupla
2 printf("Alo mundo\n");
```

- \n nova linha e fim de linha
- Salvar como "primeiro.c"
- Compilar
  - **gcc primeiro.c -o primeiro**
  - gcc é o compilador:
    - ★ Tradução do Código Fonte → código objeto
    - ★ Linguagem de Alto Nível → Linguagem de Máquina
    - ★ Efetua a análise léxica (palavras) e sintática (estrutura gramatical) e análise semântica (faz sentido - se pode ser executado) do código-fonte
- Executar
  - **./primeiro**
- Assembly:
  - gcc primeiro_programa.c -S
  - cat primeiro_programa.s

Função **printf()**

- Pode ser utilizado para **imprimir na tela valores**
- **Imprimir mensagens com valores de variáveis**:

```c
int X = 3;
int Y = 2;

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

- Tudo que está entre aspas dupla será impresso na tela
- **Especificadores de formato**: tipo do dado que será substituído pelo valor da variável
- **man 3 printf**

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

Função **printf()**

```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 "meu_programa.c"
- Compilar: gcc meu_programa.c -o meu_programa
- Executar: ./meu_programa

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

int main() {
    printf("Tamanho do float : %ld\n", sizeof(float));
    printf("FLT_MAX      :   %e\n", (float) FLT_MAX);
    printf("FLT_MIN      :   %e\n", (float) FLT_MIN);
    printf("-FLT_MAX     :   %e\n", (float) -FLT_MAX);
    printf("-FLT_MIN     :   %e\n", (float) -FLT_MIN);
    printf("DBL_MAX      :   %e\n", (double) DBL_MAX);
    printf("DBL_MIN      :   %e\n", (double) DBL_MIN);
    printf("-DBL_MAX     :   %e\n", (double) -DBL_MAX);
    printf("Precisão do float: %d\n", FLT_DIG );
    printf("Precisão do double: %d\n", DBL_DIG );

    float i = 3.402824e+38;
    float t = 3.402823e+38;
    float r = 3.402822e+38;
    printf("\n%f\n%f\n%f\n", i, t, r);

    double d = 1.797693e+308;
    printf("\n%0.151f\n", d);
    return 0;
}
```

Roteiro

Função **scanf**

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

Função **scanf** - variável x endereço

- Cada variável possui um endereço na memória
- 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 **scanf**

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**

```c
#include <stdio.h>

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

    //Inicializacao de variaveis pela entrada padrao
    scanf("%f %d", &y, &x);

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

```c
scanf("%d %d",&a,&b);
          //^ nao pode ter espaco entre o ultima entrada
    e o fecha aspas
```

Função **scanf**

- Leiam o manual das funções: **man scanf**
- Lê a entrada até consumir todos os tipos indicados na função
- Espaços em branco, tab e nova linha serão "ignorados"
- Por exemplo:
  - No código: scanf("%d %d",&a,&b);
  - Esta entrada irá funcionar:
    - 23 <enter>
    - 34 <enter>
  - Esta entrada irá funcionar: 23 34 <enter>
- Um caractere deverá corresponder **exatamente** o próximo caractere de entrada. Exemplo:
  - No código: **scanf("Numero %d", &a);**
  - Esta entrada irá funcionar: **Numero 23** <enter>
  - Esta entrada não irá funcionar:
    23 <enter>, ou esta, **Num 23** <enter>

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** - formatando a entrada

- Lendo um número fixo de inteiro: %*w*d

```c
#include <stdio.h>
int main()
{
    int x,y;
    scanf("%3d %2d",&x,&y);
    printf("%d %d\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
- 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