

Linguagem C

Variáveis e expressões

Sérgio Campos -- scampos@dcc.ufmg.br

Algoritmos

- Para resolver um problema no computador é necessário que ele seja primeiramente descrito de uma forma clara e precisa.
 - O conceito de algoritmo é frequentemente ilustrado pelo exemplo de uma receita


Algoritmos

- Um algoritmo pode ser definido como uma sequência de instruções para solucionar um problema.
- Essa sequência de instruções deve ser
 - Finita
 - Não pode ser ambígua
 - Cada instrução do algoritmo deve ser precisamente definida, sem permitir mais de uma interpretação de seu significado

Algoritmo: Bolo de Chocolate

- Aqueça o forno a 180° C
- Unte uma forma redonda
- Numa taça
 - Bata
 - 75g de manteiga
 - 250g de açúcar
 - até ficar cremoso
 - Junte
 - 4 ovos, um a um
 - 100g de chocolate derretido
 - Adicione aos poucos 250g de farinha peneirada
- Deite a massa na forma
- Leve ao forno durante 40 minutos

Algoritmo: Ambiguidade

- Em culinária é comum ter receitas com:
 - Sal a gosto
 - Bata os ovos até atingir o ponto...
 - Açúcar o quanto baste...
- E tem aquela da mulher do programador:
 - Pediu ao marido, vá à padaria e traga pães,
 - se houver ovos, traga 12.
 - Na volta ele trouxe... 12 pães... 

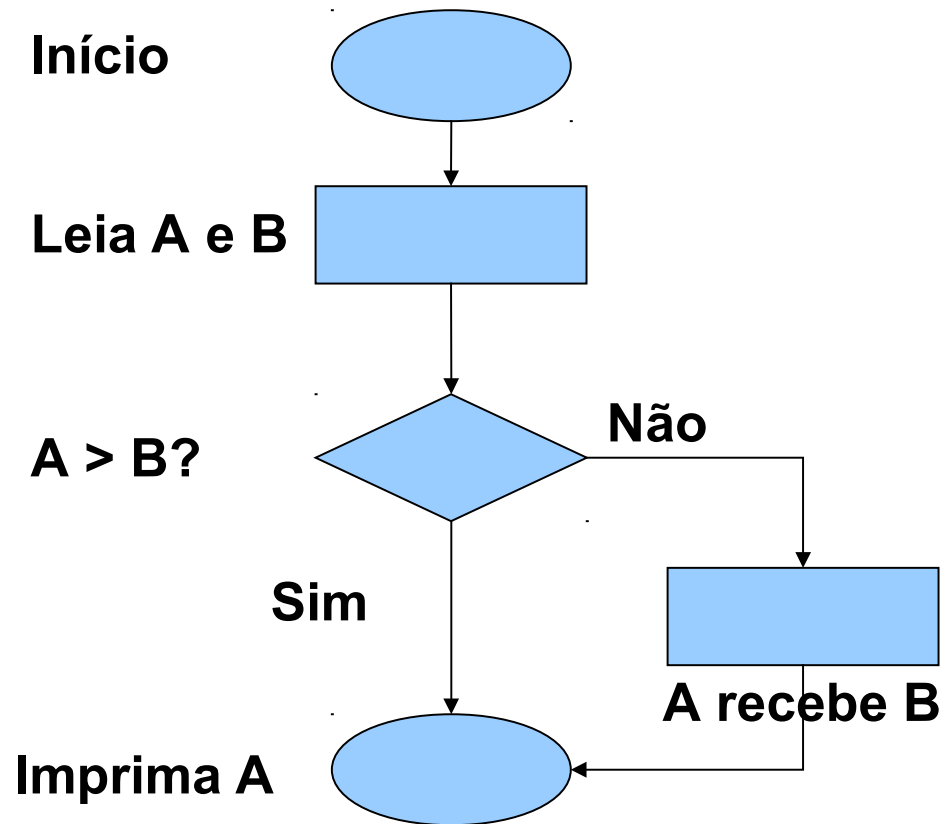
Algoritmos

- O algoritmo é a lógica do nosso problema.
 - É a sequência de passos que eu faço desenvolvo (na cabeça ou no papel) antes de escrever o programa
 - Podem existir vários algoritmos diferentes para resolver o mesmo problema

Pseudo-código e Fluxograma

- Ex.: imprimir maior valor

Leia A;
Leia B;
Se $A > B$ então
Imprima A;
Senão
Imprima B;
Fim Se



Definições

- Para resolver um problema de computação é preciso escrever um **texto**.
- Este texto, como qualquer outro, obedece **regras de sintaxe**.
- Estas regras são estabelecidas por uma **linguagem de programação**.
- Este texto é conhecido como:

Programa

Definições

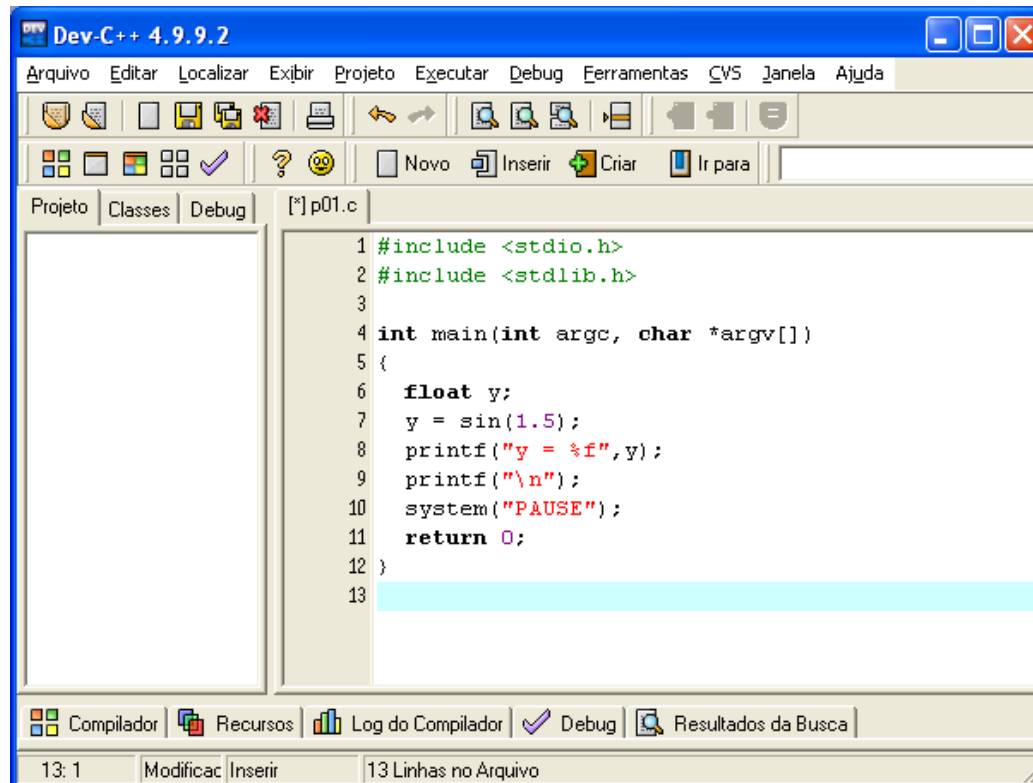
- Neste curso, será utilizada a **linguagem C**.
- A **linguagem C** é subconjunto da **linguagem C++** e, por isso, geralmente, os **ambientes de programação** da linguagem C são denominados ambientes C/C++.
- Um **ambiente de programação** contém:
 - Editor de programas: viabiliza a escrita do programa.
 - Compilador: verifica se o texto digitado obedece à sintaxe da linguagem de programação e, caso isto ocorra, traduz o texto para uma sequência de instruções em **linguagem de máquina**.



Código binário

Definições

- Que ambiente de programação iremos utilizar?
- Existem muitos, por exemplo: Microsoft Visual C++, Borland C++ Builder, Code::Blocks e DEV-C++.



Linguagens de programação

- Linguagem de Máquina
 - Computador entende apenas pulsos elétricos
 - Presença ou não de pulso
 - 1 ou 0
- Tudo no computador deve ser descrito em termos de 1's ou 0's (binário)
 - Difícil para humanos ler ou escrever
 - 00011110 = 30

Linguagens de programação

- Linguagens de Alto Nível
 - Programas são escritos utilizando uma linguagem parecida com a linguagem humana
 - Independente da arquitetura do computador
 - Mais fácil programar
 - Uso de compiladores

Compilador

- Porque o compilador traduz o programa escrito na linguagem de programação para a linguagem de máquina?

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

int main(int argc, char *argv[])
{
    float y;
    y = sin(1.5);
    printf("y = %f", y);
    printf("\n");
    system("PAUSE");
    return 0;
}
```



Compilador



```
0101010110100010011
1000101010111101111
1010100101100110011
0011001111100011100
0101010110100010011
1000101010111101111
1010100101100110011
0011001111100011100
```

- Os computadores executam instruções que estejam escritas na forma de códigos binários.
- Um programa em linguagem de máquina é chamado de programa executável.

Antigamente...

- FORTRAN (FORmula TRANslation)
 - Em 1950, um grupo de programadores da IBM liderados por John Backus produz a versão inicial da linguagem;
 - Primeira linguagem de alto nível;
- Várias outras linguagens de alto nível foram criadas
 - Algol-60, Cobol, Pascal, etc

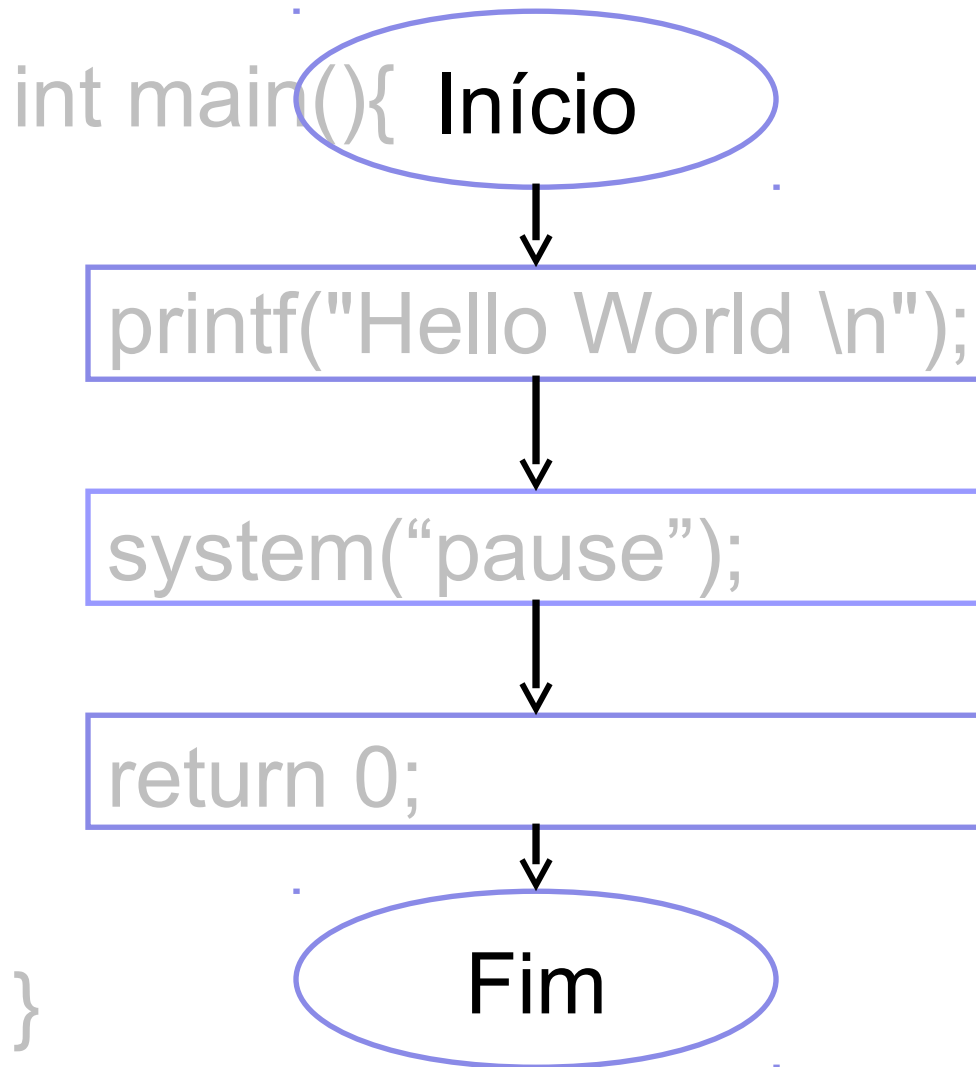
Linguagem C

- Uma das mais bem sucedidas foi uma linguagem chamada C
 - Criada em 1972 nos laboratórios por Dennis Ritchie
 - Revisada e padronizada pela ANSI em 1989
 - Padrão mais utilizado

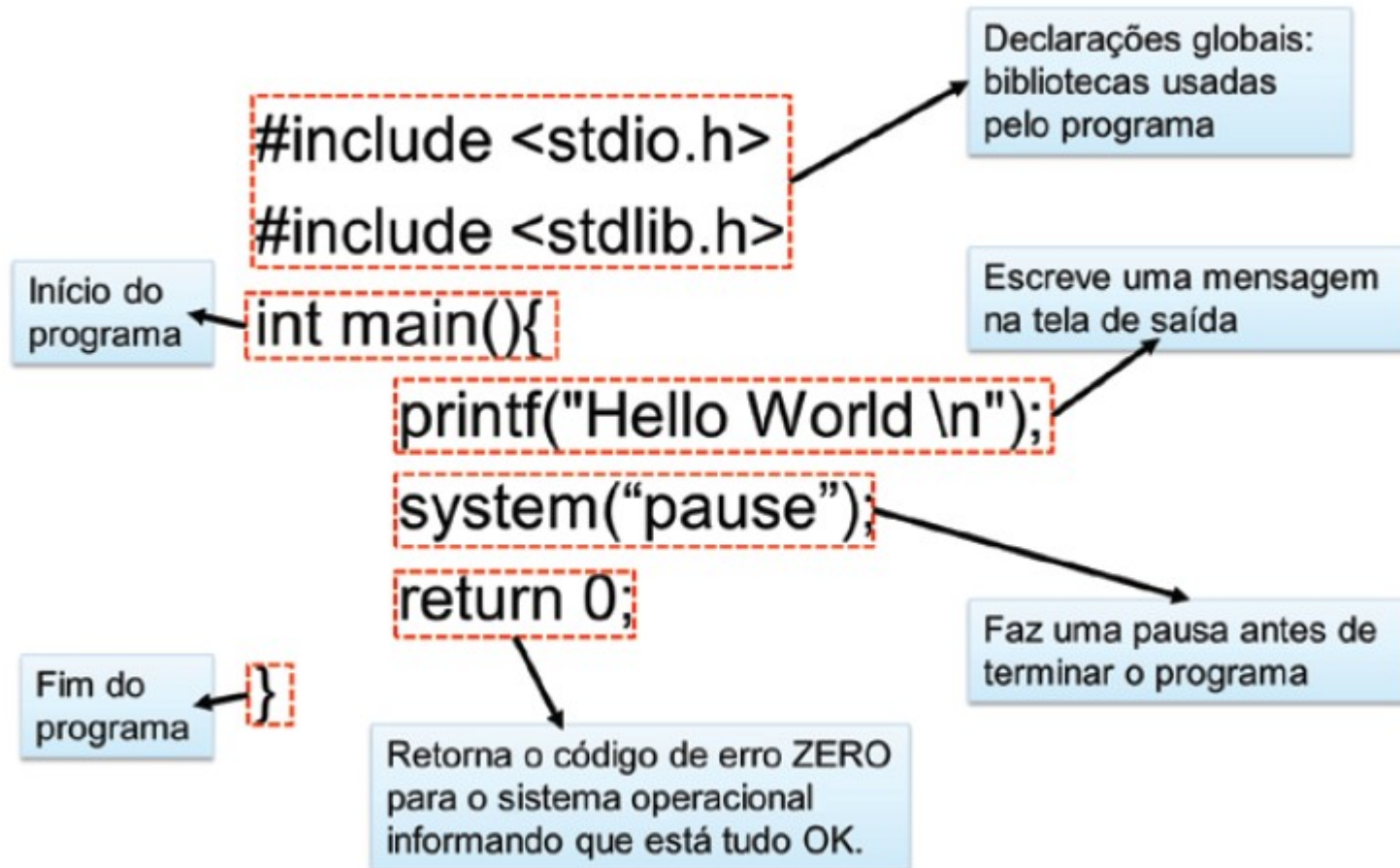
Primeiro programa em C

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    printf("Hello World \n");
    system("pause");
    return 0;
}
```


Primeiro programa em C



Primeiro programa em C



Primeiro programa em C

- Por que escrevemos programas?
 - Temos dados ou informações que precisam ser processados
 - Esse processamento pode ser algum cálculo ou pesquisa sobre os dados de entrada
 - Desse processamento, esperamos obter alguns resultados (Saídas)

Comentários

- Permitem adicionar uma descrição sobre o programa. São ignorados pelo compilador.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      /*
05       A funcao printf()
06       serve para
07       escrever na tela
08       */
09      printf("Hello World \n");
10      //faz uma pausa no programa
11      system("pause");
12      return 0;
13  }
```

Comentários

- Para que servem comentários ?
 - Para *documentar* o programa
 - Explicar o que foi feito - e como
 - Ler código sem comentários
 - Gasta 3 vezes mais tempo
 - Para se entender 1/3 do código

Comentários

```
#include <stdio.h>
int main() {
    int n, i;
    printf("Entre um numero inteiro maior que um: ");
    scanf("%d", &n);
    for (i = 2; i <= n / 2; ++i) {
        if (n % i == 0) {
            printf("Não é primo!\n");
            exit(0);
        }
    }
    printf("É primo!\n");
}
```

Comentários

```
/* Programa para calcular se um número é primo ou não.
 *
 * Inspirado em
 * https://www.programiz.com/c-programming/examples/prime-number
 *
 * Adaptado por Sérgio Campos -- 07/2020
 *
 * Este programa calcula se um número n é primo ou não verificando
 * se o número digitado é divisível ou não por todos os números
 * entre 2 e n/2.
 *
 * Entrada: Um número inteiro maior que 1
 * Saída: Uma mensagem dizendo se o número é ou não primo
 */

#include <stdio.h>
int main() {
```

Comentários

```
#include <stdio.h>
int main() {
    /* n é o número digitado; i são os números a serem testados */
    int n, i;
    printf("Entre um numero inteiro maior que um: ");
    scanf("%d", &n);
    /* calcular divisibilidade entre 2 e n/2 um por um */
    for (i = 2; i <= n / 2; ++i) {
        /* i é o próximo valor a ser testado */
        if (n % i == 0) {
            /* n DIV i = 0 => n é divisível por i e não é primo */
            printf("Não é primo!\n");
            exit(0); /* That's all folks! Game over */
        }
    }
    /* Testou todos os valores entre 2 e n/2 e não foi
       divisível, então é primo! */
    printf("É primo!\n");
}
```


Variáveis

- Na matemática
 - é uma entidade capaz de representar um valor ou expressão;
 - Pode representar um número ou um conjunto de números
 - $f(x) = x^2$

Variáveis

- Na computação
 - Posição de memória que armazena uma informação
 - Pode ser modificada pelo programa
 - Deve ser definida antes de ser usada
 - Tipo_variável lista_de_variáveis
 - Case sensitive

Variáveis

- Propriedades
 - Nome
 - Pode ter um ou mais caracteres
 - Nem tudo pode ser usado como nome
 - Tipo
 - Conjunto de valores aceitos

Variáveis

- Nome
 - Deve iniciar com letras ou underscore(_);
 - Caracteres devem ser letras, números ou underscores;
 - Palavras chaves não podem ser usadas como nomes;
 - Letras maiúsculas e minúsculas são consideradas diferentes

Variáveis

- Lista de palavras chaves

**auto break case char const
continue default do double else
enum extern float for goto if int
long register return short
signed sizeof static struct switch
typedef union unsigned void
volatile while**

Variáveis

- Quais nomes de variáveis estão corretos:
 - Contador contador1
 - comp! .var
 - Teste_123 _teste
 - int int1
 - 1contador -x
 - Teste-123 x&

Variáveis

- Corretos:
 - Contador, contador1, Teste_123, _teste, int1
- Errados
 - comp!, .var, int, 1contador, -x, Teste-123, x&

Variáveis

- Tipo
 - Define os valores que ela pode assumir e as operações que podem ser realizadas com ela
- Exemplo
 - tipo **int** recebe apenas valores inteiros
 - tipo **float** armazena apenas valores reais

Tipos básicos em C

- **char**: um byte que armazena o código de um caractere do conjunto de caracteres local
 - ***caracteres sempre ficam entre 'aspas simples'!***
 - `char c = 'a';`
- **int**: um inteiro cujo tamanho depende do processador, tipicamente 16 ou 32 bits
 - `int n = 5;`

Tipos básicos em C

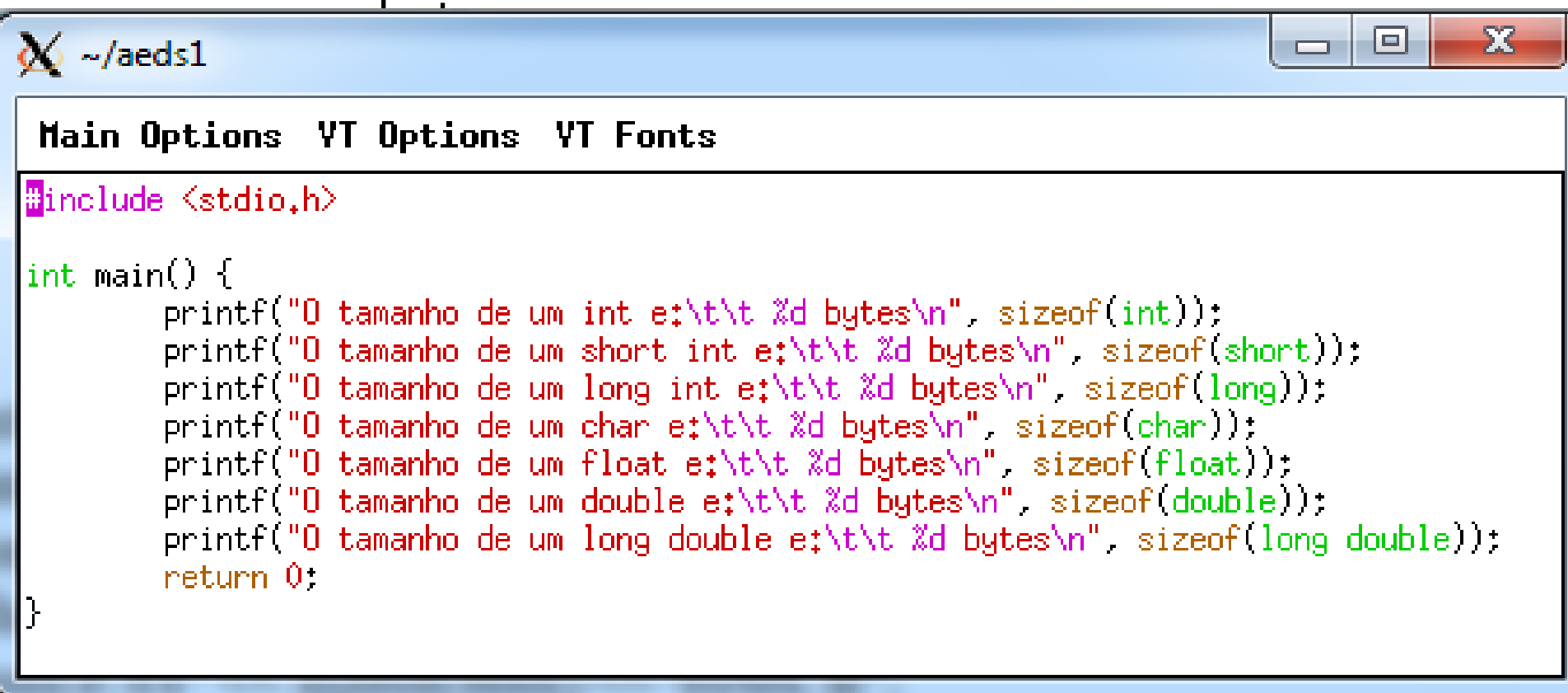
- **float**: um número real com precisão simples
 - *parte decimal usa ponto e não vírgula!*
 - float f = 5.25;
- **double**: um número real com precisão dupla
 - *Pode-se escrever números float e double usando notação científica*
 - double x = 5.0e10;

Variáveis

Tipo	Bytes	Escala
char	1	-128 a 127
int	4	-2.147.483.648 a 2.147.483.647
short	2	-32.765 a 32.767
long	4	-2.147.483.648 a 2.147.483.647
unsigned char	1	0 a 255
unsigned	4	0 a 4.294.967.295
unsigned long	4	0 a 4.294.967.295
unsigned short	2	0 a 65.535
float	4	$3,4 \times 10^{-38}$ a $3,4 \times 10^{38}$
double	8	$1,7 \times 10^{-308}$ a $3,4 \times 10^{308}$
long double	10	$3,4 \times 10^{-4932}$ a $3,4 \times 10^{4932}$
void	0	nenhum valor

Tipos e tamanhos

- Variam de acordo com o computador (ARGH!!!!)
- Use a função sizeof() para descobrir o tamanho



```
X ~/aeds1
Main Options  VT Options  VT Fonts
#include <stdio.h>

int main() {
    printf("O tamanho de um int e:\t\t %d bytes\n", sizeof(int));
    printf("O tamanho de um short int e:\t\t %d bytes\n", sizeof(short));
    printf("O tamanho de um long int e:\t\t %d bytes\n", sizeof(long));
    printf("O tamanho de um char e:\t\t %d bytes\n", sizeof(char));
    printf("O tamanho de um float e:\t\t %d bytes\n", sizeof(float));
    printf("O tamanho de um double e:\t\t %d bytes\n", sizeof(double));
    printf("O tamanho de um long double e:\t\t %d bytes\n", sizeof(long double));
    return 0;
}
```

Representação de números inteiros

- Existem várias maneiras de representar números inteiros no sistema binário.
- Forma mais simples é a **sinal-magnitude**:
 - O bit mais significativo corresponde ao sinal e os demais correspondem ao valor absoluto do número.
- **Exemplo**: considere uma representação usando cinco **dígitos binários** (ou **bits**).

<u>Decimal</u>	<u>Binário</u>
-----------------------	-----------------------

+5	00101
----	-------

-3	10011
----	-------

Desvantagens:

- Duas notações para o zero (+0 e -0).
- A representação dificulta os cálculos.

Representação de números inteiros

- Outra representação possível, habitualmente assumida pelos computadores, é a chamada **complemento-de-2**:
 - Para números positivos, a representação é idêntica à da forma sinal-magnitude.
 - Para os números negativos, a representação se dá em dois passos:
 1. Inverter os bits 0 e 1 da representação do número positivo;
 2. Somar 1 ao resultado.
 - Exemplo:

<u>Decimal</u>	<u>Binário</u>	
+6	00110	
-6	11001	(bits invertidos)
	1	(somar 1)
	11010	

Complemento de 2

Decimal	Binário s/ sinal	Binário (Compl. 2)
-8	-	1000
-7	-	1001
-6	-	1010
-5	-	1011
-4	-	1100
-3	-	1101
-2	-	1110
-1	-	1111
0	000	0000
1	001	0001
2	010	0010
3	011	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111

Números de ponto flutuante

- Números de ponto flutuante são os números reais que podem ser representados no computador.
- Ponto flutuante não é um ponto que flutua no ar!
- Exemplo:
 - Representação com ponto fixo: 12,34.
 - Representação com ponto flutuante: $0,1234 \times 10^2$.
- Ponto Flutuante ou Vírgula Flutuante?
- A representação com ponto flutuante segue padrões internacionais (IEEE-754 e IEC-559).

Comando de saída

- **printf()**
- Comando que realiza a impressão dos dados do programa

- printf("tipo de saída", lista de variáveis)

 ← printf("texto **%tipo_de_saída** texto" **expressão**);

The diagram illustrates the printf function syntax. A terminal window icon is shown on the left. An arrow points from the terminal to the function call. The function call is: printf("texto **%tipo_de_saída** texto" **expressão**);. The format specifier **%tipo_de_saída** and the expression **expressão** are highlighted with red dashed boxes. A bracket above the string connects the format specifier to the expression, indicating that the expression's value is formatted according to the specifier.

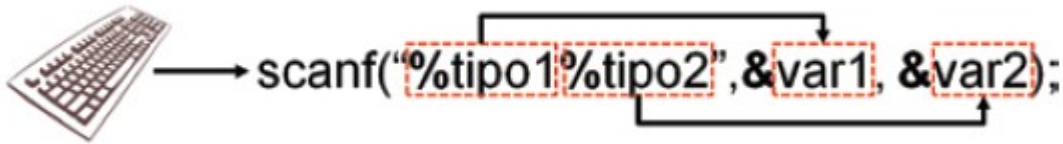
- Alguns tipos de saída
 - %c – escrita de **um** caractere
 - %d – escrita de números inteiros
 - %f – escrita de número reais
 - %s – escrita de **vários** caracteres

Comando de saída

- Ex:
 - Saída de um único valor inteiro
`printf("%d",x);`
 - Saída de mais de um único valor
`printf("%d%d",x,y);`
`printf("%d %d",x,y);`
- No tipo de saída, pode-se formatar toda a saída
 - `printf("Total = %d",x+y);`

Comando de entrada

- **scanf()**
- Comando que realiza a leitura dos dados de entrada
 - scanf("tipo de entrada", lista de variáveis)



- Alguns “tipos de entrada”
 - `%c` – leitura de **um** caractere
 - `%d` – leitura de números inteiros
 - `%f` – leitura de número reais
 - `%s` – leitura de **vários** caracteres

Comando de entrada

- Ex:
 - Leitura de um único valor
`int x;`
`scanf("%d",&x);`
 - Leitura de mais de um valor
`int x,y;`
`scanf("%d%d",&x,&y);`
- Obs: na leitura de vários valores, separar com espaço.

Comando de entrada

- **getchar()**

- Comando que realiza a leitura de um único caractere

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      char c;
05      c = getchar();
06      printf("Caractere: %c\n", c);
07      printf("Codigo ASCII: %d\n", c);
08      system("pause");
09      return 0;
10  }
```

Constantes

- Como uma variável, uma constante também armazena um valor na memória do computador.
- Entretanto, esse valor não pode ser alterado: é constante.
- Para constantes é obrigatória a atribuição do valor.

Constantes

- Usando **#define**

- Você deverá incluir a diretiva de pré-processador **#define** antes de início do código:

```
#define PI 3.1415
```

- Usando **const**

- Usando **const**, a declaração não precisa estar no início do código.

```
const double pi = 3.1415;
```

Constantes char

- A linguagem C utiliza vários códigos chamados códigos de barra invertida.

Código	Comando
\a	som de alerta (bip)
\b	retrocesso (backspace)
\n	nova linha (new line)
\r	retorno de carro (carriage return)
\v	tabulação vertical
\t	tabulação horizontal
\'	apóstrofe
\"	aspa
\\	barra invertida (backslash)
\f	alimentação de folha (form feed)
\?	símbolo de interrogação
\0	caractere nulo (cancela a escrita do restante)

Constantes char

01	#include <stdio.h>
02	#include <stdlib.h>
03	int main(){
04	printf("Hello World\n");
05	printf("Hello\nWorld\n");
06	printf("Hello \\World\n");
07	printf("\"Hello World\"\n");
08	system("pause");
09	return 0;
10	}

Saída	Hello World Hello World Hello \ World "Hello World"
-------	---

Erros de sintaxe

- Atenção!
 - O **programa executável** só será gerado se o texto do programa não contiver **erros de sintaxe**.
 - Exemplo: considere uma **string**. Uma **sequência de caracteres delimitada por aspas**.
 - Se isso é uma string e se tivéssemos escrito:

```
printf("y = %f, y);
```

- O compilador iria apontar um erro de sintaxe nesta linha do programa e exibir uma mensagem tal como:

```
undetermined string or character constant
```

Erros de lógica

```
#include <stdio.h>
int main() {
    /* n é o número digitado; i são os números a serem testados */
    int n, i;
    printf("Entre um numero inteiro maior que um: ");
    scanf("%d", &n);
    /* calcular divisibilidade entre 2 e n/2 um por um */
    for (i = 2; i <= n / 2; ++i) {
        /* i é o próximo valor a ser testado */
        if (n % i == 0) {
            /* n DIV i = 0 => n é divisível por i e não é primo */
            printf("Não é primo!\n");
            exit(0); /* That's all folks! Game over */
        }
    }
    /* Testou todos os valores entre 2 e n/2 e não foi
       divisível, então é primo! */
    printf("É primo!\n");
}
```

Erros de lógica

```
#include <stdio.h>
int main() {
    /* n é o número digitado; i são os números a serem testados */
    int n, i;
    printf("Entre um numero inteiro maior que um: ");
    scanf("%d", &n);
    /* calcular divisibilidade entre 2 e n/2 um por um */
    for (i = 3; i <= n / 2; ++i) {
        /* i é o próximo valor a ser testado */
        if (n % i == 0) {
            /* n DIV i = 0 => n é divisível por i e não é primo */
            printf("Não é primo!\n");
            exit(0); /* That's all folks! Game over */
        }
    }
    /* Testou todos os valores entre 2 e n/2 e não foi
       divisível, então é primo! */
    printf("É primo!\n");
}
```

Erros de lógica

- Um resultado foi gerado, mas ele não é correto.
- Se um programa executável não produz os resultados corretos, é porque ele contém erros de lógica ou bugs.

Depuração

- O processo de identificação e correção de erros é denominado **depuração (debug)**:
 - Erros de sintaxe: escrito errado!
 - `a = b // 2; /* oops, era b / 2 */`
 - O compilador pega, fácil
 - Erros de lógica: não era isto que eu queria dizer!
 - `sqrt(b*b + 4*a*c) /* oops era - */`
 - O compilador não pega, gera um resultado que *parece* correto.