



TÉCNICAS DE PROGRAMAÇÃO

Aula 01 – Programação Estruturada em C++



Objetivos de Aprendizagem

1. Descrever o paradigma de programação estruturada.

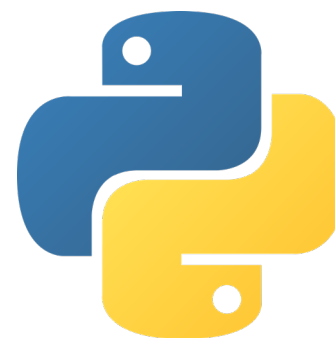


Por que estudar C/C++?

- C é uma linguagem de programação de uso geral que existe há quase 50 anos.
- C foi usado para escrever tudo, desde sistemas operacionais a programas complexos, como:
 - Interpretador Python
 - Git
 - Banco de Dados Oracle
 - Microsoft Windows
 - Arduino

Por que estudar C/C++?

- A versatilidade de C/C++ é por design.
 - É uma linguagem de baixo nível que se relaciona intimamente com o modo como as máquinas funcionam e ainda assim é fácil de aprender.





Quiz

- C/C++ é um:
 - a. Linguagem de script do lado do cliente
 - b. Linguagem de programação de uso geral
 - c. Programa de edição de fotos



Programa em C

- Um programa em C inicia com a inclusão de bibliotecas.
- Uma biblioteca é um arquivo que contém comandos complementares, que são utilizados pelo nosso programa.
- Para a inclusão de uma biblioteca devemos usar um comando que é chamado de **diretiva de compilação**.
- Este comando informa ao compilador quais bibliotecas devem ser anexadas ao programa executável.
- Assim, o comando para a inclusão de uma biblioteca tem a seguinte sintaxe:

```
#include <nome da biblioteca>
```



Programa em C

- De maneira geral, os arquivos de biblioteca têm a terminação `.h` (esse `h` vem de *header*, ou seja, *este é um arquivo de cabeçalho*).
- *Para cada* biblioteca, que será incluída no nosso programa, devemos colocar um comando `#include`.
- *Na medida em que formos aprendendo* os comandos, serão indicadas quais bibliotecas devem ser incluídas no programa.

`#include <stdio.h>`



Programa em C

- A quantidade de bibliotecas e quais bibliotecas serão incluídas dependem dos comandos que serão utilizados no programa.
- Pode ser que em um programa não seja necessário incluir nenhuma biblioteca.



Hello World!

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

Para usar a função **printf**, precisamos primeiro incluir o arquivo de cabeçalho.

```
int main() {  
    printf("Olá, mundo!\n");  
    return 0;  
}
```

A função **main ()** é o ponto de entrada para um programa.



Tipos de dados

- C/C++ suporta os seguintes tipos de dados básicos:
 - **int**: número inteiro, um número inteiro.
 - **float**: ponto flutuante, um número com uma parte fracionária.
 - **double**: valor de ponto flutuante de precisão dupla.
 - **char**: caractere único.



Tipos de dados

- A quantidade de armazenamento necessária para cada um desses tipos varia de acordo com a plataforma.
- C/C++ possui um operador de tamanho interno que fornece os requisitos de memória para um tipo de dados específico.



Exemplo

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

```
int main() {  
    printf("int: %d \n", sizeof(int));  
    printf("float: %d \n", sizeof(float));  
    printf("double: %d \n", sizeof(double));  
    printf("char: %d \n", sizeof(char));  
  
    return 0;  
}
```



E cadê o tipo de dado booleano?

- Observe que C/C++ não tem um tipo booleano.
- Uma instrução **printf** pode ter vários especificadores de formato com argumentos correspondentes para substituir os especificadores.
 - Especificadores de formato também são chamados de especificadores de conversão.
- Aprenderemos mais sobre especificadores de formato nos próximos slides.



Quiz

- Quais dos seguintes opções são tipos de dados válidos em C/C++?
 - a. int, float, string, char
 - b. int, duplo, char, booleano
 - c. int, flutuante, duplo, char
 - d. int, bool, string



Variáveis

- Uma variável é um nome para uma área na memória.
- O nome de uma variável (também chamada de identificador) deve começar com uma letra ou um sublinhado e pode ser composto de letras, dígitos e o caractere de sublinhado.
 - `nome;`
 - `_nome;`
 - `numMatricula;`



Variáveis

- As convenções de nomenclatura variável diferem, no entanto, é comum o uso de letras minúsculas com um sublinhado para separar palavras (snake_case).
- As variáveis também devem ser declaradas como um tipo de dados antes de serem usadas.
 - `int numMatricula;`
 - `float salario;`



Variáveis

- O valor para uma variável declarada é alterado com uma instrução de atribuição.
- Por exemplo, as seguintes instruções declaram uma variável inteira `minha_var` e, em seguida, atribui o valor 42:

```
int minha_var;  
minha_var = 42;
```



Variáveis

- Você também pode declarar e inicializar (atribuir um valor inicial) uma variável em uma única instrução:

```
int minha_var = 42;
```



Exemplo

```
#include <stdio.h>
int main() {
    int a = 8, b = 34;
    float salario = 56.23;
    char letra = 'Z';
    int c = a+b;
    printf("%d \n", c);
    printf("%f \n", salario);
    printf("%c \n", letra);

    return 0;
}
```



Tome nota: C é *case-sensitive*!

- A linguagem de programação C faz distinção entre maiúsculas e minúsculas, portanto:
 - **minha_Var** e **minha_var** são dois identificadores diferentes.



Constantes

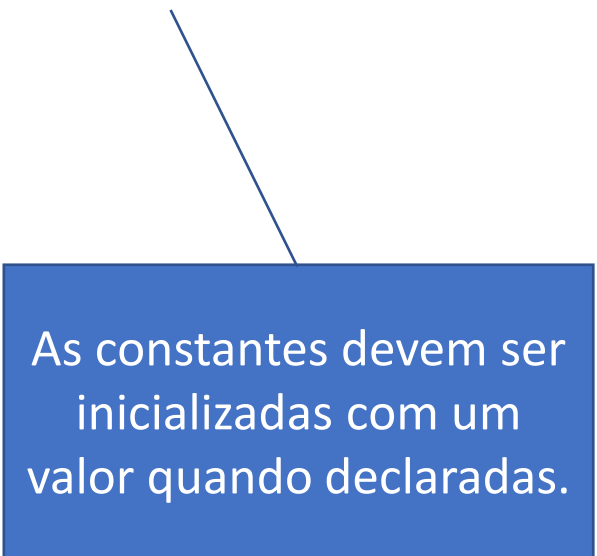
- Uma constante armazena um valor que não pode ser alterado a partir de sua atribuição inicial.
- Usando constantes com nomes significativos, o código fica mais fácil de ler e entender.
- Para distinguir constantes de variáveis, uma prática comum é usar identificadores em maiúsculas.



Exemplo (const)

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

```
int main() {  
    const double PI = 3.14;  
    printf("%f", PI);  
  
    return 0;  
}
```



As constantes devem ser inicializadas com um valor quando declaradas.

Exemplo (#define)

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

```
#define PI 3.14
```

```
int main() {  
    printf("%f", PI);  
    return 0;  
}
```

NÃO coloque um caractere ponto-e-vírgula no final das instruções #define. Este é um erro comum.



Constantes

- Antes da compilação, o pré-processador substitui cada identificador de macro no código pelo valor correspondente da diretiva.
 - Nesse caso, toda ocorrência de PI é substituída por 3.14.
- O código final enviado ao compilador já terá os valores constantes no lugar.



Operadores aritméticos

- C/C++ suporta operadores aritméticos:
 - + (adição)
 - - (subtração)
 - * (multiplicação)
 - / (divisão)
 - % (divisão de módulo)



Operadores aritméticos

- Os operadores geralmente são usados para formar uma expressão numérica como $10 + 5$, que neste caso contém dois operandos e o operador de adição.
- Expressões numéricas são frequentemente usadas em instruções de atribuição.



Exemplo

```
#include <stdio.h>
int main() {
    int largura = 10;
    int comprimento = 5;
    int area;

    area = largura * comprimento;
    printf("%d \n", area);  /* 50 */

    return 0;
}
```



Divisão

- C/C++ tem dois operadores de divisão: / e %.
- O operador divisão / executa de maneira diferente, dependendo dos tipos de dados dos operandos.
- Quando os dois operandos são tipos de dados **int**, a divisão inteira, também chamada divisão truncada, remove qualquer restante para resultar em um inteiro.



Divisão

- Quando um ou ambos os operandos são números reais (flutuante ou duplo), o resultado é um número real.
- O operador % retorna apenas o restante da divisão inteira.
- É útil para muitos algoritmos, incluindo a recuperação de dígitos de um número.



Exemplo

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

```
int main() {  
    int i1 = 10;  
    int i2 = 3;  
    int quociente, resto;  
    float f1 = 4.2;  
    float f2 = 2.5;
```



Exemplo

```
float resultado;
```

```
quociente = i1 / i2; // 3
```

```
resto = i1 % i2; // 1
```

```
resultado = f1 / f2; // 1.68
```

```
return 0;
```

```
}
```



Operador de Precedência

- C/C++ avalia uma expressão numérica com base na precedência do operador.
- Os $+$ e $-$ são iguais em precedência, assim como $*$, $/$ e $\%$.
- Os $*$, $/$ e $\%$ são executados primeiro na ordem da esquerda para a direita e depois em $+$ e $-$, também na ordem da esquerda para a direita.



Operador de Precedência

- Você pode alterar a ordem das operações usando parênteses () para indicar quais operações devem ser executadas primeiro.
- Por exemplo, o resultado de **$5 + 3 * 2$ é 11**, onde o resultado de **$(5 + 3) * 2$ é 16**.



Exemplo

```
#include <stdio.h>

int main() {
    int a = 6;
    int b = 4;
    int c = 2;
    int resultado;
    resultado = a - b + c; // 4
    printf("%d \n", resultado);
}
```



Exemplo

```
resultado = a + b / c; // 8  
printf("%d \n", resultado);
```

```
resultado = (a + b) / c; // 5  
printf("%d \n", resultado);
```

```
return 0;  
}
```

Operadores de Precedência

- C/C++ pode não avaliar uma expressão numérica conforme desejado quando a propriedade associativa permite qualquer ordem.
- Por exemplo, $x * y * z$ pode ser avaliado como $(x * y) * z$ ou como $x * (y * z)$.
- Se a ordem for importante, divida a expressão em instruções separadas.



Operadores de Atribuição

- Uma instrução de atribuição avalia a expressão no lado direito do sinal de igual primeiro e depois atribui esse valor à variável no lado esquerdo de =
- Isso possibilita o uso da mesma variável nos dois lados de uma declaração de atribuição, o que é frequentemente feito na programação.



Exemplo

- `int x = 3;`
- `x = x + 1; /* x is now 4 */`
- `x += 1; /* x = x + 1 */`



Exemplo

```
int x = 2;
```

```
x += 1; // 3
```

```
x -= 1; // 2
```

```
x *= 3; // 6
```

```
x /= 2; // 3
```

```
x %= 2; // 1
```

```
x += 3 * 2; // 7
```

Incremento e Decremento

- A adição de 1 a uma variável pode ser feita com o operador de incremento `++`.
- Da mesma forma, o operador de decremento `--` é usado para subtrair 1 de uma variável
- Exemplo:

```
z--; /* decrementa z em 1 */
```

```
y++; /* incrementa y em 1 */
```




Incremento e Decremento

- Os operadores de incremento e decremento podem ser usados como
 - **Pré-fixado** (antes do nome da variável)
 - **Pós-fixado** (após o nome da variável)
- O modo como você usa o operador pode ser importante em uma declaração de atribuição, como no exemplo a seguir.



Exemplo

```
z = 3;
```

```
x = z--;
```

```
/* atribui 3 para x,
```

```
** então decrementa z em 2 */
```

```
y = 3;
```

```
x = ++y;
```

```
/* incrementa y em 4,
```

```
** então atribui 4 para x */
```



Comentários

- Comentários são informações explicativas que você pode incluir em um programa para beneficiar o leitor do seu código.
- O compilador ignora comentários, portanto eles não afetam um programa.



Comentário de bloco

- Um comentário começa com uma barra asterisco `/`
`*` e termina com uma barra asterisco
`*` `/` e pode estar em qualquer lugar no seu código.
- Os comentários podem estar na mesma linha que uma declaração ou podem abranger várias linhas.



Exemplo

```
#include <stdio.h>

/* Um programa simples em C
 * Versão 1.0
 */
int main() {
    /* Imprime uma string */
    printf("Olá, mundo!");
    return 0;
}
```



Comentários de linha única

- O C ++ introduziu um comentário com barra dupla `//` como uma maneira de comentar linhas únicas.
- Alguns compiladores C também suportam esse estilo de comentário.



Exemplo

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

```
int main() {  
    int x = 42; //int para um número inteiro  
  
    //%d é substituído por x  
    printf("%d", x);  
  
    return 0;  
}
```



Conversão de tipos

- Quando uma expressão numérica contém operandos de diferentes tipos de dados, eles são automaticamente convertidos conforme necessário em um processo chamado conversão de tipo.
- Por exemplo, em uma operação envolvendo ***floats*** e ***ints***, o compilador converterá os valores ***int*** em valores flutuantes.



Exemplo

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

```
int main() {
```

```
    float preco = 6.50;
```

```
    int aumento = 2;
```

```
    float novo_preco;
```

```
    novo_preco = preco + aumento;
```

```
    printf("Novo preço é %4.2f", novo_preco);
```

```
    /* Saída: Novo preço é 8.50 */
```

```
    return 0;
```

```
}
```



Exemplo

```
float media;  
int total = 23;  
int contador = 4;  
  
media = (float) total / contador;  
/* media = 5.75 */
```



Conversões de tipos

- A conversão explícita de tipo, mesmo quando o compilador pode fazer a conversão automática de tipo, é considerada um bom estilo de programação.




Entrada (Input)

- C suporta várias maneiras de receber a entrada do usuário.
- **getchar** () Retorna o valor da próxima entrada de caractere único.
- **scanf** () verifica a entrada que corresponde aos especificadores de formato.
- C++ possui o operador **cin >>** para receber a entrada do usuário.



Entrada (Input)

- A função **gets** () é usada para ler a entrada como uma sequência ordenada de caracteres, também chamada de *string*.
- Uma *string* é armazenada em uma matriz de caracteres.



Exemplo (getchar)

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

```
int main() {
```

```
    char a = getchar();
```

```
    printf("Você digitou: %c", a);
```

```
    return 0;
```

```
}
```



Exemplo (gets)

```
#include <stdio.h>

int main() {
    char a[100];

    gets(a);

    printf("Você digitou: %s", a);

    return 0;
}
```



Exemplo (scanf)

```
#include <stdio.h>

int main() {
    int a;
    scanf("%d", &a);

    printf("Você digitou: %d", a);

    return 0;
}
```




Exemplo (scanf)

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

```
int main() {  
    int a, b;  
    printf("Entre com dois números:");  
    scanf("%d %d", &a, &b);  
  
    printf("\nSoma: %d", a+b);  
  
    return 0;  
}
```




Entrada (Input)

- O sinal **&** antes do nome da variável é o operador de endereço.
- Fornece o endereço ou localização na memória de uma variável.
- Isso é necessário porque **scanf** () coloca um valor de entrada em um endereço variável



Saída (Output)

- Já usamos a função **printf** () para gerar saída nas lições anteriores.
- Agora, abordamos várias outras funções que podem ser usadas para saída.
- **putchar** () gera um único caractere.
- A função **puts** () é usada para exibir a saída como uma *string*.
 - Uma *string* é armazenada em uma matriz de caracteres.
- Em C++ temos o operador **cout <<** que gera saída para o monitor do usuário.



Exemplo (putchar)

```
#include <stdio.h>

int main() {
    char a = getchar();

    printf("Você digitou: ");
    putchar(a);

    return 0;
}
```



Exemplo (puts)

```
#include <stdio.h>
int main() {
    char a[100];

    gets(a);

    printf("Você digitou: ");
    puts(a);

    return 0;
}
```



Exemplo (printf)

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

```
int main() {
```

```
    int a;
```

```
    scanf("%d", &a);
```

```
    printf("Você digitou: %d", a);
```

```
    return 0;
```

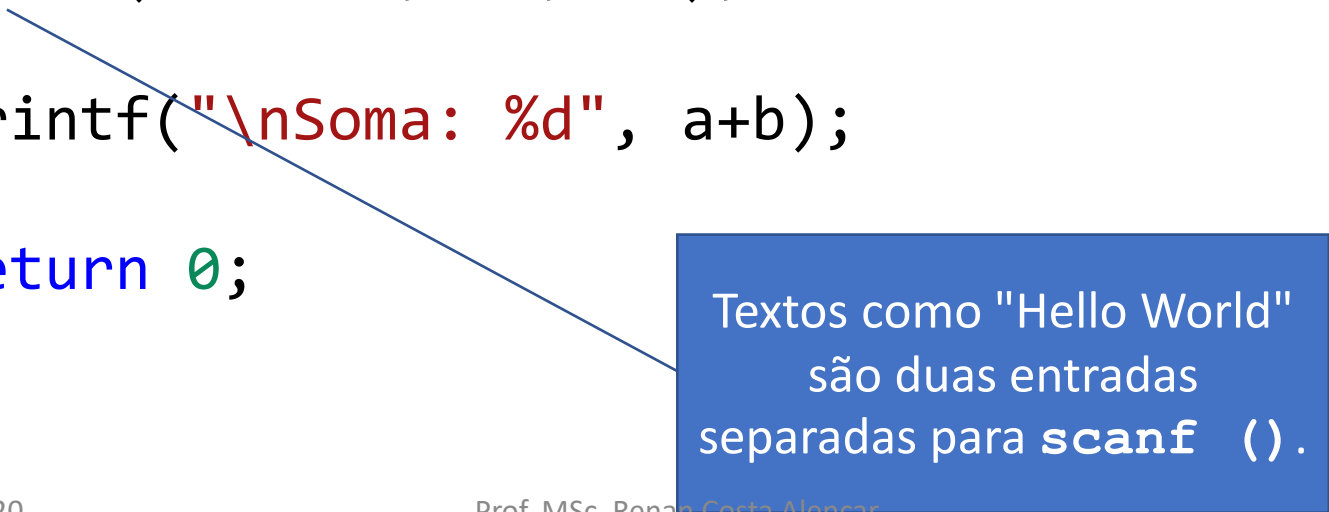
```
}
```



Exemplo (scanf)

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

```
int main() {  
    int a, b;  
    printf("Entre com dois números:");  
    scanf("%d %d", &a, &b);  
  
    printf("\nSoma: %d", a+b);  
  
    return 0;  
}
```



Textos como "Hello World"
são duas entradas
separadas para `scanf` ().



Palavras Reservadas

- As linguagens de programação são formadas por um conjunto de regras de sintaxe e semântica que ditam como o programa deve ser escrito.
- Com isso, dentro dessas regras, existe um conjunto de palavras que tem um significado para a linguagem de programação – são as **palavras reservadas**.
- Uma palavra reservada é, essencialmente, um comando e, na maioria das vezes, as palavras reservadas de uma linguagem definem o que pode ser feito e como pode ser feito.



Palavras Reservadas

- As palavras reservadas são de uso exclusivo da gramática da linguagem, por isso, não podem ser utilizadas, pelo programador, para dar nome a alguma variável, constante ou função do seu programa.
- Assim, um programador não pode ter uma variável chamada “int” no seu programa C/C++, já que “int” é uma palavra reservada que indica um tipo de dado.
- Na linguagem C/C++ temos 32 palavras reservadas. Todas as palavras reservadas do C são escritas em minúsculo.
- A tabela abaixo mostra as palavras reservadas, conforme definido pelo padrão ANSI, para a linguagem C/C++.





Palavras Reservadas

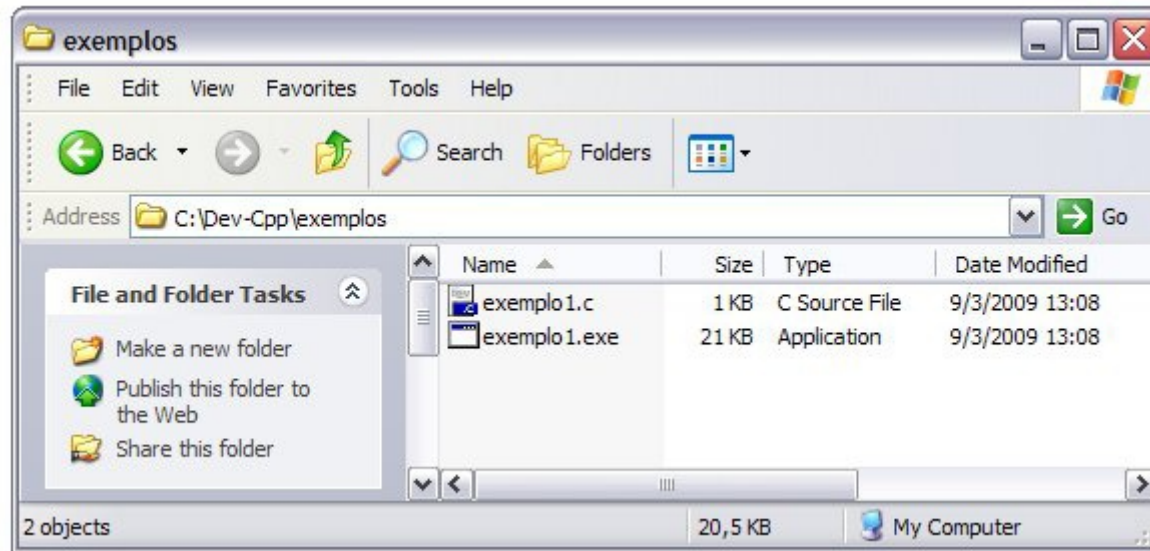
auto	default	float	unsigned
break	do	for	return
case	double	goto	short
char	else	union	signed
const	typedef	if	sizeof
switch	while	int	static
volatile	enum	long	struct
continue	extern	register	void



Os fundamentos do ambiente C

- No Dev-cpp, o programa executável é armazenado no mesmo diretório que estiver armazenado o código fonte.
- Existe um diretório contendo o arquivo do código fonte (.c – *C source file*) e o programa executável (.exe – *application*).
- *O arquivo do programa executável não pode ser editado*, o que nós podemos editar é o arquivo do código fonte.

Os fundamentos do ambiente C/C++

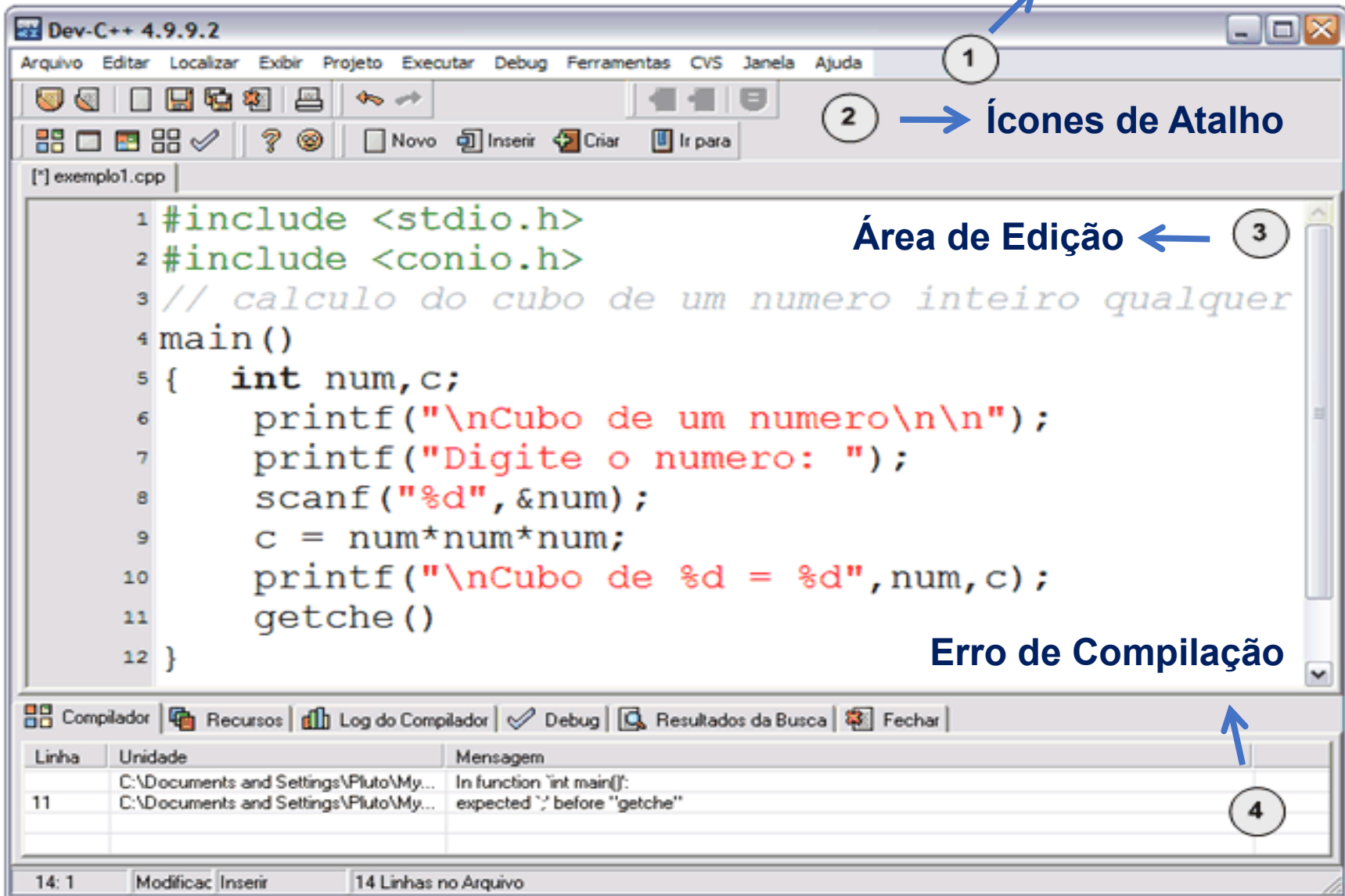




Conhecendo o Dev-C++

- O Dev-Cpp é um ambiente de programação que edita, compila e executa programas C e C++.
- Sabem porque iremos usar o Devcpp para desenvolver nossos programas?
 - O Dev-C++ é gratuito (muito importante!) e possui uma interface bastante amigável (mais importante ainda!).

Menu Principal





Conhecendo o Dev-C++

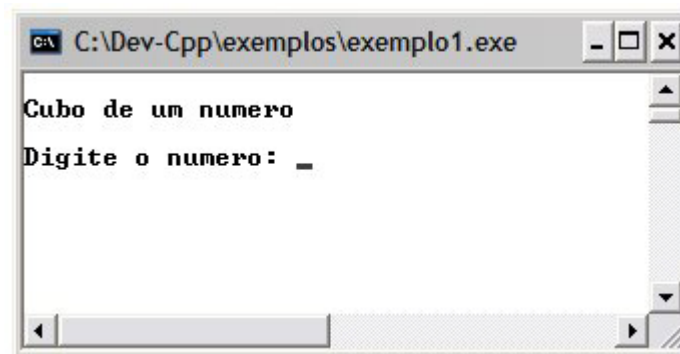
- Mesmo que vocês tenham instalado o Dev-cpp em Português, as mensagens de erro são apresentadas em Inglês.



Conhecendo o Dev-C++

- Segue abaixo um roteiro do que vocês precisam fazer para editar, compilar e executar seus programas no Dev-C++:
 1. Crie um arquivo fonte novo na opção: Arquivo/Novo/Arquivo Fonte;
 2. Digite o programa fonte na área de edição. Ao terminar de editá-lo, salve o arquivo;
 3. Compile o programa na opção: Executar/Compilar;
 4. Se der algum erro no programa, vejam as indicações de erro fornecidas pelo compilador. Conserte os erros, salve o arquivo e compile novamente. Isto deve ser feito até que seja apresentada uma mensagem indicando que o programa não tem erros de compilação.
 5. Se vocês acessarem o diretório que o arquivo do código fonte foi armazenado, notarão que foi criado um arquivo com a extensão .exe (com o mesmo nome do arquivo do código fonte). Este é programa executável. Para executá-lo, escolha a opção Executar/Executar no Dev-C++. Imediatamente, aparecerá a janela de execução do programa.

Conhecendo o Dev-C++





Exercício

1. Quando é que precisamos incluir uma biblioteca em um programa C/C++?
2. O que é diretiva de compilação?
3. O que deve ter no corpo do programa principal?
4. O que é uma palavra reservada?
5. Por que devemos comentar nossos programas?