

Algoritmos e Linguagens de Programação

PE-02 – v1.1

Prof. Paulo Joia Filho

- 1 Introdução
- 2 Algoritmos
- 3 Linguagens de Programação
- 4 Conclusões

- 1 **Introdução**
 - Objetivos
 - Motivação
- 2 Algoritmos
- 3 Linguagens de Programação
- 4 Conclusões

Objetivos

Os principais objetivos desta aula são:

- Introduzir a ideia de algoritmo.
- Discutir os passos necessários para construção de algoritmos.
- Apresentar conceitos sobre linguagens de programação.
- Apresentar uma visão geral da linguagem C.

Motivação

Por que estudar algoritmos?

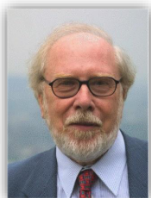
“A noção de algoritmo é básica para toda a programação de computadores”

- ❖ KNUTH - Professor da Universidade de Stanford, autor da coleção “The art of computer programming”.



“O conceito central da programação e da ciência da computação é o conceito de algoritmo”

- ❖ WIRTH - Professor da Universidade de Zurique, autor de diversos livros na área e responsável pela criação de linguagens de programação como PASCAL.



1 Introdução

2 Algoritmos

- Conceitos básicos
- Representação
- Passos para a construção
- Problemas que pode resolver
- Construção de algoritmos


3 Linguagens de Programação

4 Conclusões

Algoritmos

Definição

Algoritmo é a descrição de uma sequência de passos que deve ser seguida para a realização de uma tarefa.

- Ao contrário do que se pensa, o conceito de algoritmo não foi criado para atender os requisitos da computação.
 -  A programação de computadores é apenas um dos campos de aplicação dos algoritmos.
- Em computação:
 - É utilizado para representar a solução de um problema;
 - Serve como uma linguagem intermediária entre a linguagem humana e as linguagens de programação.

Importante

Para resolver um problema em um computador, em geral, é necessário um algoritmo.

Lembrando que:

- Para a mesma tarefa podem existir vários algoritmos.
- Em algumas situações é interessante encontrar o algoritmo mais eficiente, o qual apresenta **menor complexidade de tempo** e/ou uso de memória.

Exemplos clássicos (não computacionais) para fins de aprendizagem:

- Trocar uma lâmpada.
- Fazer um bolo.
- Trocar o pneu do veículo.

...muitos outros!

Representação de um algoritmo

As formas mais comuns de representação de um algoritmo são:
Descrição narrativa, Diagrama de blocos e Pseudocódigo.

Ambas consistem em analisar o enunciado do problema e escrever os passos a serem seguidos para sua resolução utilizando...

❑ Descrição narrativa

- ▶ Uma linguagem natural (por exemplo, a língua portuguesa).

❑ Diagrama de blocos (ou Fluxograma)

- ▶ Símbolos gráficos predefinidos.

❑ Pseudocódigo (ou Portugal)

- ▶ Um conjunto de regras predefinidas.

Exemplo de algoritmo

Multiplicação de dois números

Algoritmo em descrição narrativa:

Passo 1 — Receber dois números que serão multiplicados.










Passo 2 — Multiplicar os números.

Passo 3 — Mostrar o resultado obtido na multiplicação.

Algoritmo em pseudocódigo:

```
inicio  
real: N1, N2, M  
escreva("Digite dois números: ")  
leia(N1, N2)  
M ← N1 * N2  
escreva("Multiplicação = ", M)  
fim
```

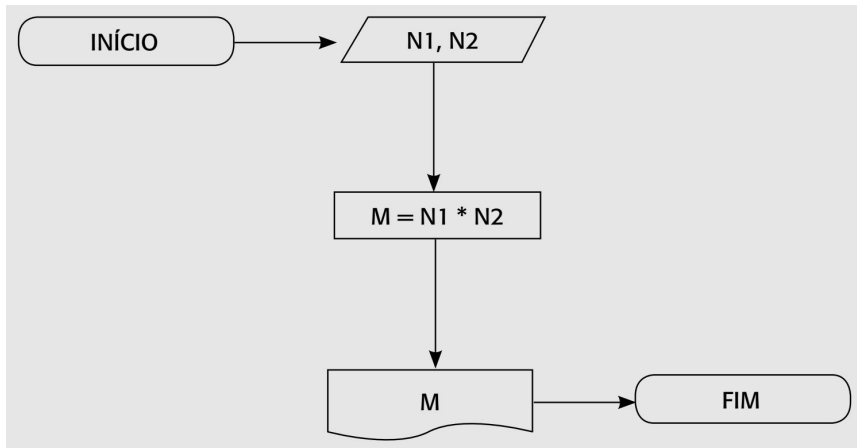
Diagrama de Blocos – Alguns Símbolos

	Terminal. Símbolo utilizado para indicar o início e o fim do algoritmo.
	Fluxo de dados. Permite indicar o sentido do fluxo de dados. Serve para conectar os símbolos ou blocos existentes.
	Processamento. Representa a execução de operações lógicas ou matemáticas e atribuição de valores.
	Entrada e saída de dados. Operações de entrada e saída de dados (genérica).
	Entrada manual. Representa a entrada manual de dados, normalmente efetuada por um teclado conectado ao computador.
	Exibição da saída. Utilizado para exibir informações ou resultados. Alguns autores empregam o símbolo à esquerda para exibição em vídeo e o da direita para saída de dados em impressora.
	Decisão. Símbolo utilizado para indicar uma tomada de decisão, apontando a possibilidade de desvios para diversos outros pontos de fluxo, dependendo do resultado de comparações.
	Conector de diagramas. Utilizado quando é preciso particionar o diagrama. Quando ocorre mais de uma partição, coloca-se uma letra ou nº dentro do símbolo de conexão para identificar os pares de ligação.
	Conector de página. Específico para indicar conexão do fluxo em outra página.

Exemplo de algoritmo

Multiplicação de dois números

Algoritmo usando diagrama de blocos:



Exemplo Ilustrativo

Comum em Publicações Científicas

[Joia et al., 2015]

Algoritmo *Column Selection Method (CSM)*

Entrada: Matriz de dados A , parâmetro k e número de colunas c .**Saída:** Matriz C com c colunas selecionadas.**procedimento** *RepresentativeSelection*(A, k, c)1: $[U, \Sigma, V^T] \leftarrow \text{SVDTRUNCADO}(A, k)$ 2: **para toda** coluna j em V^T **faça**3: $\pi_j \leftarrow \sum_{i=1}^k (v_j^i)^2$ 4: **fim para**5: $Idx \leftarrow$ índices das colunas com os c maiores valores de π 6: $C \leftarrow A(Idx)$

Comparação (da representação):

```
início
real: N1, N2, M
escreva("Digite dois números: ")
leia(N1, N2)
M ← N1 * N2
escreva("Multiplicação = ", M)
fim
```

Vantagens e desvantagens das formas de representação de algoritmos...

● Descrição narrativa

- ✓ Não é necessário aprender nenhum conceito novo, pois a linguagem natural já é bem conhecida.
- ✗ A linguagem natural abre espaço para várias interpretações, dificultando sua transcrição para programa.

● Diagrama de blocos

- ✓ Elementos gráficos são mais simples de entender do que texto.
- ✗ É necessário aprender a simbologia dos fluxogramas.
- ✗ O algoritmo resultante não apresenta muitos detalhes, dificultando sua transcrição para programa.

● Pseudocódigo

- ✓ A passagem do algoritmo para qualquer linguagem de programação é quase imediata, bastando conhecer as palavras reservadas da linguagem a ser utilizada.
- ✗ É necessário aprender as regras do pseudocódigo.

Passos para a construção de algoritmos

- 1 Compreender completamente o problema a ser resolvido.
 - ▶ Destacar os pontos mais importantes, e
 - ▶ Os objetos que o compõem.
- 2 Definir os dados de entrada.
 - ▶ Quais dados serão fornecidos?
- 3 Definir os procedimentos.
 - ▶ Quais cálculos serão efetuados?
 - ▶ Quais as restrições do problema?
- 4 Definir os dados de saída.
 - ▶ Quais dados serão gerados após o processamento?
- 5 Construir o algoritmo utilizando uma das representações conhecidas.
- 6 Testar o algoritmo realizando simulações.

Que problemas pode resolver?

Inúmeros problemas podem ser resolvidos por meio de algoritmos.

Alguns exemplos são:

- Ordenação de uma lista de valores.
- Cálculo do fatorial de um número.
- Sortear aleatoriamente um número em um intervalo dado, ...
- Cálculo estrutural em vigas.
- Cálculos estatísticos, como média, mediana e desvio padrão.
- Cálculo da correlação de dados.

Muitos outros!

Desafio

Situação-problema

Em um sistema de correio eletrônico, como separar *emails* legítimos de *spams*?

Análise do problema:

- **Entrada**

- ▶ *Email* (no caso mais simples, um arquivo de caracteres).

- **Saída**

- ▶ Sim/Não indicando se a mensagem é um *spam* ou não.

- **Algoritmo**

- ▶ Decisão guiada pelos dados, ao invés de seguir simples instruções programadas.
- ▶ Problema que não existia até a década de 80.
- ▶ Novos esforços em *Ciência da Computação*.



Uma ciência em contínua evolução...

Possível solução para o problema do *spam*:

- A partir de mensagens de exemplo, “*aprender*” o que constitui um *spam*.
- A “*máquina*” (computador) extrai automaticamente o algoritmo para esta tarefa.

Existem muitas aplicações onde não temos um algoritmo bem definido, apenas exemplos de dados.

- ▶ Área conhecida como “*Aprendizado de Máquina*” ou *Machine Learning* (ML). Para mais detalhes consulte [Alpaydin, 2010](#).



Figura 1: *Machine Learning pipeline.*

Construção de algoritmos

Pré-requisitos

A construção de um algoritmo requer:

- Um número finito de passos;
- Que cada passo seja precisamente definido, sem ambiguidades;
- Uma ou mais entradas tomadas de conjuntos bem definidos;
- Uma ou mais saídas;
- Que exista uma condição de fim para quaisquer das entradas, em tempo finito.

NÃO se aprende algoritmos...

- ✗ Lendo algoritmos
- ✗ Copiando algoritmos

Aprende-se algoritmos...

- ✓ Construindo algoritmos
- ✓ Testando algoritmos

Construção de algoritmos

Praticando...

Exemplo 1

Construir um algoritmo que calcule a média aritmética de quatro notas bimestrais quaisquer fornecidas por um aluno.

Apresentar:

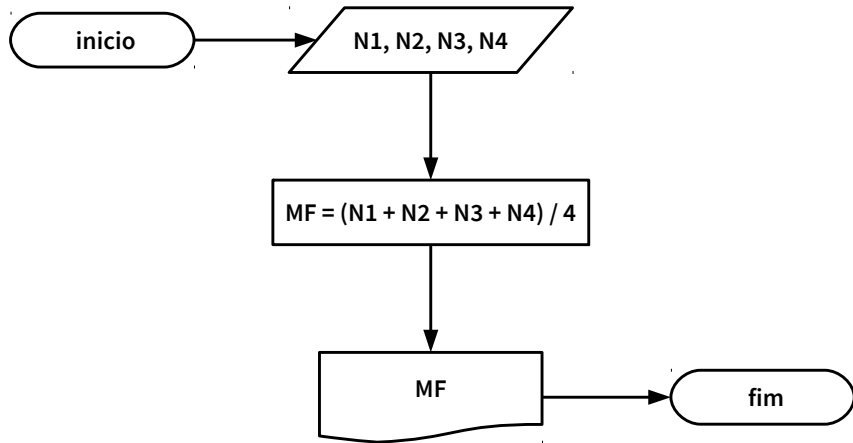
- ▶ Diagrama de blocos;
- ▶ Pseudocódigo;

Informações:

- Dados de entrada: quatro notas (N1, N2, N3 e N4);
- Dado de saída: média final (MF).

Solução

Diagrama de Blocos



Solução

Pseudocódigo

```
1  inicio // começo do algoritmo

3  // declaração das variáveis
   real: N1, N2, N3, N4, MF

5

   // mensagem para o usuário
7  escreva("Digite as quatro
   notas: ")

9

   // entrada dos dados
11 leia(N1, N2, N3, N4)

13 // processamento
   MF ← (N1 + N2 + N3 + N4) / 4

15

   // saída da informação
17 escreva("Média final = ", MF)

19 fim // término do algoritmo
```

Extra: Implementação Scilab × C

```

1  // começo do programa

3  // não é necessário declarar tipos de variáveis

5  // mensagem para o usuário
printf("Digite as quatro notas:\n");

7

9  // entrada dos dados
N1 = input("N1 = ");
N2 = input("N2 = ");
11 N3 = input("N3 = ");
N4 = input("N4 = ");

13

15 // processamento
MF = (N1 + N2 + N3 + N4)/4;

17 // saída da informação
printf("Média final = %g\n", MF);

19

21 // término do programa

```

```

1  #include <stdio.h>

3  void main() { // começo do programa

5      // declaração das variáveis
float N1, N2, N3, N4, MF;

7

9      // mensagem para o usuário
printf("Digite as quatro notas: \n");

11     // entrada dos dados
scanf("%f%f%f%f", &N1, &N2, &N3,
      &N4);

13

15     // processamento
MF = (N1 + N2 + N3 + N4) / 4;

17     // saída da informação
printf("Média final = %g\n", MF);

19

21 } // término do programa

```

Discussão do exemplo

- Comentários no código

- Para aumentar a clareza do código recomenda-se o uso de comentários;
- Neste estudo, comentários devem começar com //
- Eles servem para explicar o que está acontecendo no código;
- Seu uso é uma excelente prática.

- Armazenamento de informações

- Para que as informações sejam mantidas e utilizadas em diferentes partes do código, é necessário declará-las;
- Em outras palavras, definir seu tipo e nome;¹
- Cada elemento declarado recebe o nome de “variável”.

¹ Em Scilab não é necessário definir o tipo de dado contido em uma variável, porém, em C, tal declaração é requerida pelo compilador.

1 Introdução

2 Algoritmos

3 Linguagens de Programação

- Conceitos fundamentais
- A linguagem C

4 Conclusões

Linguagem de Programação

- Uma linguagem de programação é um método padronizado para comunicar instruções para um computador.
- É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador.

A linguagem de programação permite especificar:

- Sobre quais dados o programa vai atuar;
- Como estes dados serão armazenados ou transmitidos, e
- Quais ações devem ser tomadas sob várias circunstâncias.



Linguagens de baixo e alto nível

► Linguagem de baixo nível

- Conjunto de instruções que podem ser executadas diretamente pela CPU.

Ex: Linguagem de máquina (alfabeto=0,1), Assembly.

► Linguagem de alto nível

- Conjunto de instruções mais próxima da linguagem humana.

Ex: Pascal, Basic, C#, Python, Java, etc.

Compiladores × Interpretadores

● Compiladores

- ▶ Traduzem um programa-fonte escrito em uma linguagem de alto nível para uma linguagem de baixo nível (linguagem de máquina).

Ex: C, C++, C#, Fortran, COBOL, ...

- ▶ Vantagem: um programa compilado, em geral, executa mais rapidamente do que um programa interpretado.

● Interpretadores

- ▶ São programas que executam um programa-fonte sem a necessidade de traduzi-lo para a linguagem de máquina.

Ex: Python, Scilab, JavaScript, VBScript, Lua, PHP, Tcl, R, ...

- ▶ Vantagens de um programa interpretado:
 - Pode ser executado diretamente pelo interpretador, sem a necessidade de um passo extra de compilação;
 - Mais fácil aprendizagem;
 - Ideal para modelagem / prototipagem.

A Linguagem C

Histórico

- Origem do C
 - Dennis Ritchie
 - BCPL \rightarrow B \rightarrow C em 1970
- Com a popularidade dos microcomputadores, surgiu um grande número de implementações da linguagem C
 - compatíveis, porém com algumas discrepâncias
- Para evitar discrepâncias, o *American National Standards Institute* (ANSI), em 1983, criou o padrão C ANSI
 - Modificado em 1989 (C89)
 - Modificado em 1999 (C99) \leftarrow *versão utilizada neste curso!*
 - Modificado em 2011 (C11)

Características da Linguagem C

- Estruturada
 - Porém, não estruturada em blocos.
- Médio Nível
 - Alto nível: Pascal, Python, Java...
 - Médio nível: C/C++
 - Baixo nível: Assembly
- Para programadores
 - Foi criada, influenciada e testada por programadores.
- Outras características...
 - Portabilidade do compilador;
 - Conceito e uso de bibliotecas;
 - Acesso ao hardware quando necessário.

Palavras Reservadas

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

Forma geral de um programa C

- Declarações globais
- Funções definidas pelo programador: f_1, f_2, \dots, f_N
- Função `main`

Global declarations

```
return_type f1(parameter list) {  
    statement sequence  
}
```

```
return_type f2(parameter list) {  
    statement sequence  
}
```

...

```
return_type fN(parameter list) {  
    statement sequence  
}
```

```
int main(parameter list) {  
    statement sequence  
}
```


Exemplo de programa C (estruturado)

```
1  #include <stdio.h>
2  #include <math.h>
3
4  /* Programa para calcular a área de um círculo, dado o raio. */
5  const double PI = 3.141592653589793;
6
7  double calcular_area(double raio) {
8      return PI * pow(raio, 2);
9  }
10
11 int main(int argc, char *argv[]) {
12     double raio, area;
13     printf("Digite o raio: ");
14     if (scanf("%lf", &raio) == 0) {
15         printf("Informe um valor numérico.\n");
16         return -2; // código de erro pré-definido
17     }
18     area = calcular_area(raio);
19     printf("Area = %.3lf\n", area);
20     return 0;
21 }
```

Compilação de um Programa C

– usando gcc no Linux

- 1 Criar o programa;
- 2 Compilar o programa;
- 3 Linkeditar o programa com as funções necessárias das bibliotecas.

Sintaxe do compilador gcc:

```
$ gcc [parametros] <nome-do-arquivo>
```

Exemplo:

```
$ gcc area_circulo.c -lm -o area_circulo
```

Para executar no Linux:

```
$ ./area_circulo
```

Nota: o parâmetro -lm instrui o compilador a fazer a linkedição com a biblioteca math.

Fases da compilação

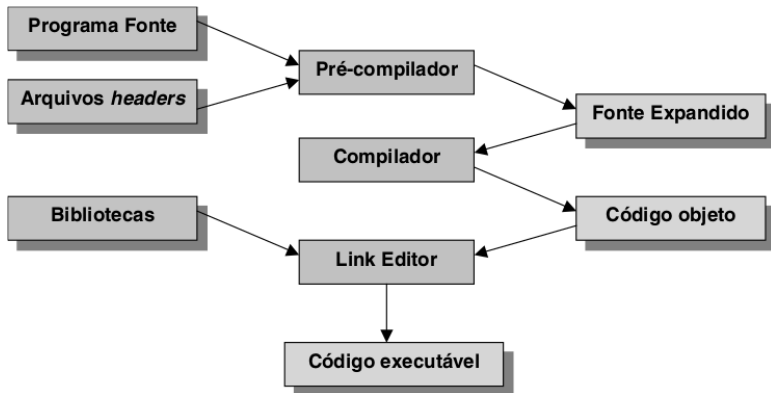


Figura: O processo de compilação (Retirado de [Laureano, 2005](#)).

O processo de compilação de um programa é constituído de três fases distintas: pré-compilação, compilação e linkedição.

Fases da compilação

● Pré-compilação

- Nesta fase, o programa fonte é lido e, caso encontre diretivas de pré-compilação, elas serão processadas.²
- O pré-compilador gera então um código intermediário que será lido pelo compilador.

● Compilação

- O compilador traduz a linguagem de código intermediário e gera o código objeto, que é o código assembly da arquitetura.

● Linkedição

- O código objeto não é executado diretamente depois de traduzido, pois o programa pode conter referências a outros programas (bibliotecas).
- O linkeditor agrupa os códigos traduzidos separadamente em um único módulo, formando assim o programa executável final.

²Diretivas de pré-compilação começam com o caractere #. Ex: #define, #if, #include.

Diretiva `#include`

Sintaxe:

```
#include <arquivo.h>
```

```
#include "arquivo.h"
```

- A diretiva `#include` indica ao pré-compilador para ler o arquivo informado e colocar seu código no programa fonte intermediário.
- O arquivo incluído geralmente possui protótipos de funções a serem utilizadas pelos programas.
- Quando um arquivo *header* é local (criado pelo programador) deve-se colocar o nome dele entre aspas, caso contrário (bibliotecas do sistema) deve-se colocar o nome dele entre `< >`.

Na prática, esta regra não precisa ser seguida, pois a utilização de aspas indica ao compilador para procurar o arquivo *header* primeiro no diretório local e depois nos diretórios do sistema. Já o `< >` indica ao compilador para procurar o arquivo *header* primeiro nos diretórios do sistema e depois no diretório local.

Mapa de memória em C

Quatro regiões logicamente distintas e com funções específicas:

- Código do Programa
- Variáveis Globais
- *Heap*
 - região de memória livre que o programa pode usar, via funções de alocação dinâmica; muitas aplicações como arrays e estruturas que mudam de tamanho dinamicamente.
- Pilha
 - endereço de retorno das chamadas de função
 - argumentos para funções
 - variáveis locais
 - armazena o estado atual da CPU



C versus C++

- C++ é uma extensão da linguagem C, projetada para suportar programação orientada a objetos.
- C++ contém toda a linguagem C e mais um conjunto de extensões orientadas a objetos.
- Quase sempre um programa C pode ser compilado usando um compilador C++.³

Quer testar?

Compile o programa de exemplo anterior, `area_circulo.c`, utilizando `g++` ao invés de `gcc`.

³A recíproca não é verdadeira.

Ambientes de Desenvolvimento Integrado

Definição

Ambientes de Desenvolvimento Integrado ou *Integrated Development Environments* (IDEs) são programas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

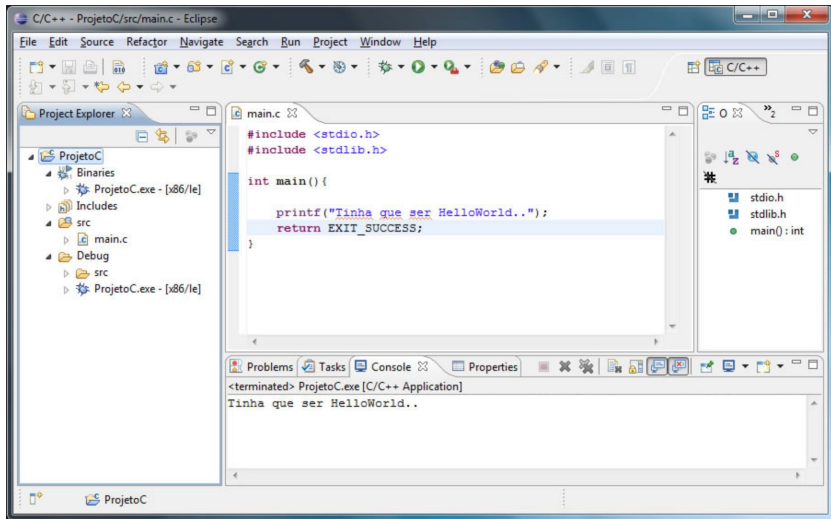
Algumas IDEs disponíveis para desenvolvimento em C/C++ são:

- ❖ **Eclipse**
 - ▶ Linux, Windows e Mac OS.
- ❖ **Dev-C++**
 - ▶ Somente Windows.
- ❖ **Anjuta**
 - ▶ Somente Linux.
- ❖ **Code::Blocks**
 - ▶ Linux, Windows e Mac OS.

Exemplo de IDE

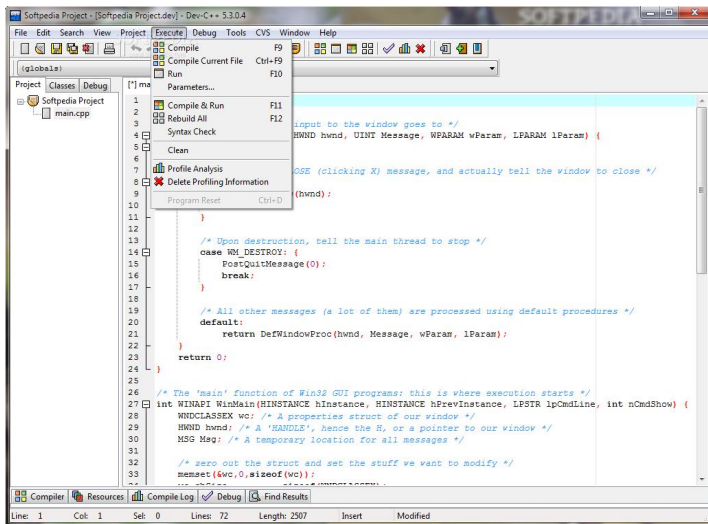


<https://eclipse.org>



Exemplo de IDE

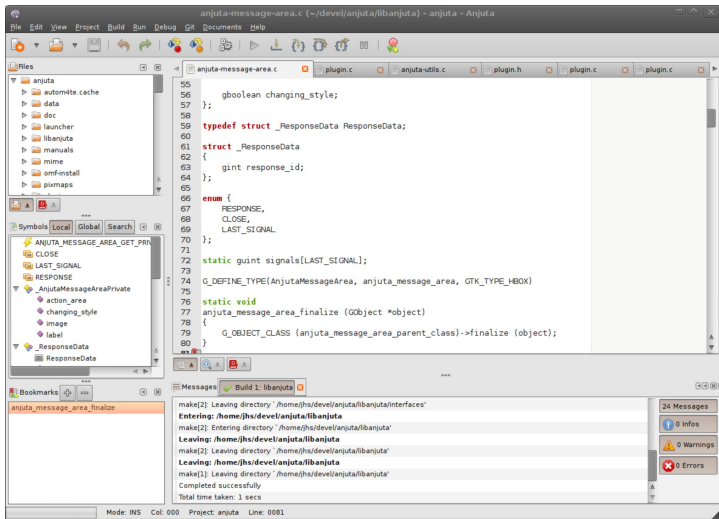
❖ **Dev-C++** <https://sourceforge.net/projects/orwelldevcpp>



Exemplo de IDE



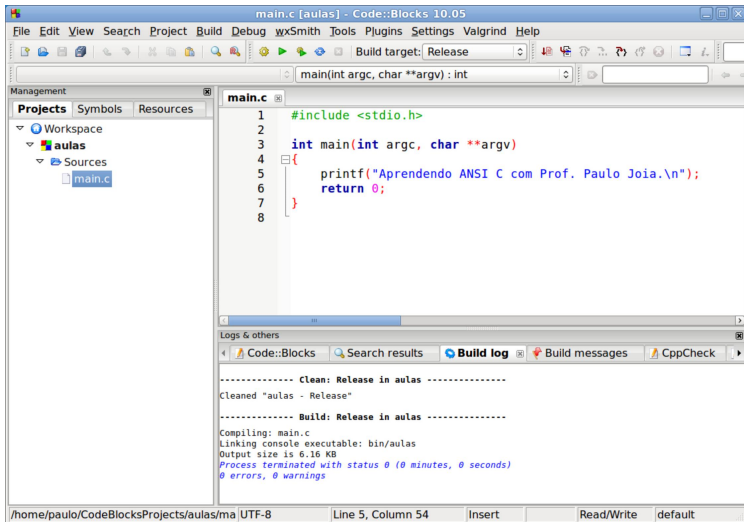
<http://anjuta.org>



Exemplo de IDE

❖ Code::Blocks

<http://www.codeblocks.org>



- 1 Introdução
- 2 Algoritmos
- 3 Linguagens de Programação
- 4 Conclusões**
 - Considerações Finais
 - Referências Bibliográficas

Considerações Finais

- ❖ Nesta aula foi apresentada a ideia de algoritmo, as formas de representá-lo e os passos para sua construção.
 - ❖ Os conceitos básicos sobre linguagem de programação, além de uma visão geral da linguagem C.
- ✎ *É importante rever todos os conceitos e praticar os exemplos em casa.*

Referências Bibliográficas I



Aguilar, L. J. (2008).

Programação em C++: Algoritmos, Estruturas de Dados e Objetos.
McGraw-Hill, São Paulo.



Alpaydin, E. (2010).

Introduction to Machine Learning.
Adaptive Computation and Machine Learning. MIT Press, Cambridge, 2nd edition.



Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2002).

Algoritmos: Teoria e Prática.
Elsevier, Rio de Janeiro.



Drozdek, A. (2009).

Estrutura de Dados e Algoritmos em C++.
Cengage Learning, São Paulo.



Forbellone, A. L. V. e Eberspacher, H. F. (2005).

Lógica de Programação: A Construção de Algoritmos e Estrutura de Dados.
Pearson Prentice Hall, São Paulo, 3 edition.



Joia, P., Petronetto, F., e Nonato, L. G. (2015).

Uncovering Representative Groups in Multidimensional Projections.
Computer Graphics Forum, 34(3):281–290.

Referências Bibliográficas II



Knuth, D. E. (2005).

The Art of Computer Programming.

Addison-Wesley, Upper Saddle River, NJ, USA.



Laureano, M. (2005).

Programando em C para Linux, Unix e Windows.

Brasport, Rio de Janeiro.



Pinheiro, F. d. A. C. (2012).

Elementos de Programação em C.

Bookman, Porto Alegre.



Sedgewick, R. (1998).

Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, Searching.

Addison-Wesley, Boston, 3rd edition.



Szwarcfiter, J. L. e Markenzon, L. (1994).

Estruturas de Dados e Seus Algoritmos.

LTC, Rio de Janeiro.



Tenenbaum, A. A., Langsam, Y., e Augenstein, M. J. (1995).

Estruturas de Dados Usando C.

Makron Books, São Paulo.