Módulo 1 - Introdução à programação

Olá mundo

```
#include <stdio.h>

// Função principal
int main(void){

    // Função printa no console
    printf("Olá mundo\n");

    // Função printa no console
    printf("Este é meu primeiro código em C");

    // Diz ao sistema que o código terminou
    return 0;

    // Aperte F9 para executar
}
```

Estrutura do C

Um programa em C vai buscar executar todos os comandos presentes em sua função principal, chamada *main*. Por isso, no início da sua aprendizagem, certifique-se de estar escrevendo seu código dentro desta função. Para entender de forma mais prática, veja o código abaixo:

```
#include <stdio.h>

int main(void){

    // Escreva seu código aqui

}
```

É uma estrutura simples. Veja que o código começa com uma instrução chamada `#include <stdio.h>` , esta instrução adiciona uma biblioteca padrão de C em seu código, veremos mais a frente o que isto significa. Logo abaixo temos nossa função *main*, ela inicia com uma declaração `int`, seguida por seu nome *main* e, entre parênteses, uma outra declaração `void`, estes conceitos também serão explicados

mais a frente. E por fim temos a presença de duas chaves { }, tudo o que for colocado dentro destas chaves diz ao programa o que deve ser executado por nosa função *main*.

Vamos escrever nosso primeiro código. Para facilitar, vamos utilizar um editor de código online pelo link https://www.onlinegdb.com/online_c_compiler. Acesse, apague o conteúdo da página e digite o código abaixo:

Ao pressionar a tecla F9 de seu teclado, o programa será compilado e um console será aberto imprimindo sua resposta.

Espaçamento: Os espaços são desconsiderados pelo compilador

Chaves e ponto e vírgula:

Comentários: Um bom programador comenta todos os seus códigos para quando ele precisar mais a frente ele entender rapidamente. Comentários de Uma e várias linhas

Algoritmos e lógica de programação

O que são algoritmos? Algoritmos são um conjunto finito de regras a serem seguidas afim de resolver um problema ou atingir um objetivo. A principal característica do algoritmo é sua ordem, se esta ordem for alterada, o resultado será outro ou pode nem funcionar. Exemplos famosos de algoritmos são: receitas de bolo, manual de instruções, contas matemáticas, entre outros. São todos sequências de passo a passo que, se forem feitos de forma errada, não atingirão um resultado satisfatório.

Vamos tomar como exemplo o ato de atravessar a rua. Se você tivesse a tarefa de mostrar a uma pessoa que nunca atravessou a rua como atravessar da forma correta, como você faria?

Modo 1

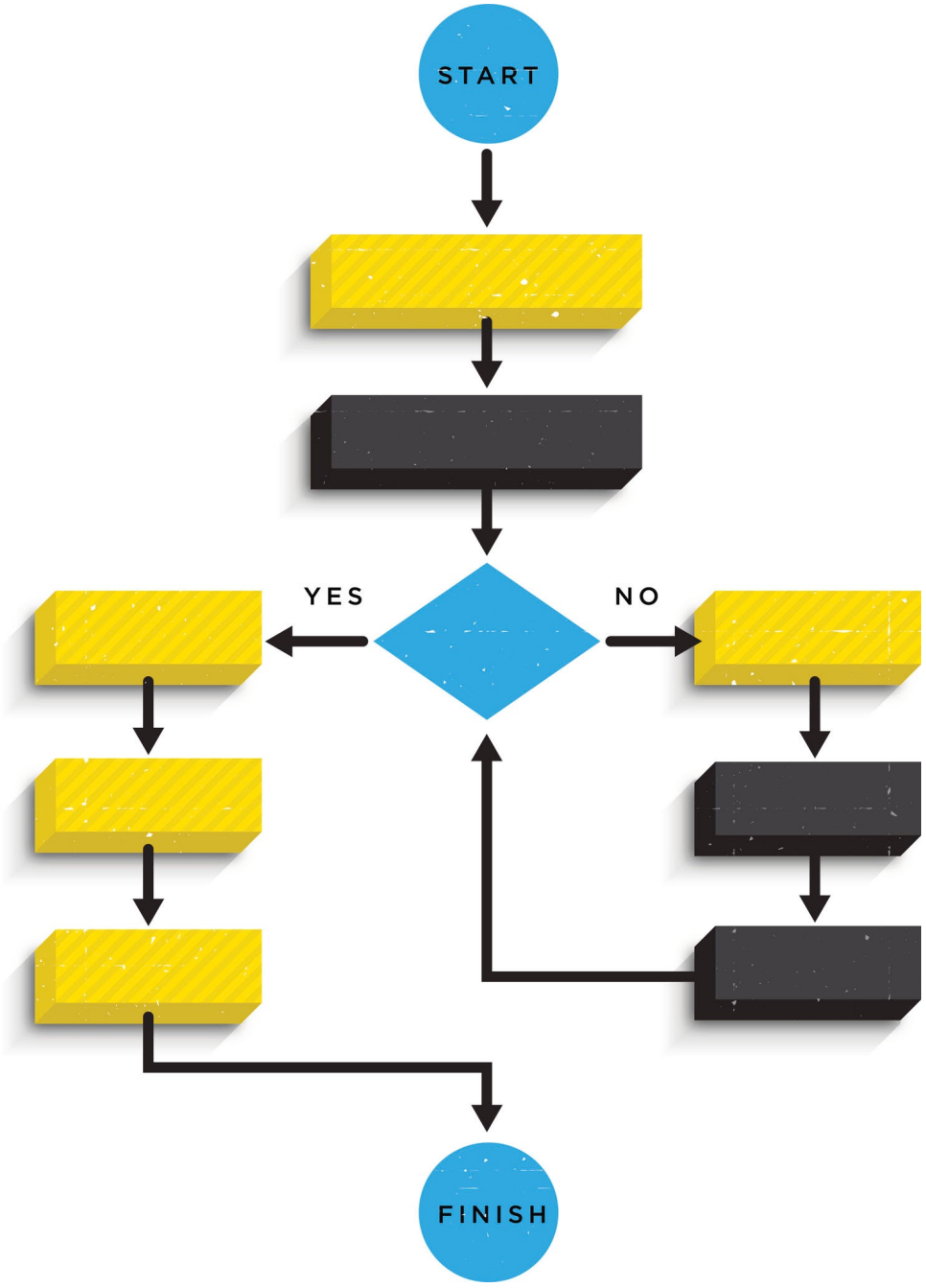
```
1- Algoritmo atravessar a rua
2-   Olhar para a direita
3-   Olhar para a esquerda
4-     Se estiver vindo carro
5-         Não atravesse
6-     Senão
7-         Atravesse
8-     Fim se
9- Fim algoritmo
```

Modo 2

```
1- Algoritmo atravessar a rua
2-   Atravesse
3-     Se estiver vindo carro
4-        Olhar para a direita
5-     Senao
6-        Olhar para a esquerda
7-     Fim se
8-   Não atravesse
9- Fim algoritmo
```

Estudar algoritmos é diferente do que estudar uma linguagem. Uma linguagem é só uma ferramenta, já algoritmos são as aplicações para essas ferramentas. Mas ainda sim é possível aprender os dois ao mesmo tempo. Portanto, neste curso utilizaremos a linguagem C para ir aprendendo a lógica de programação

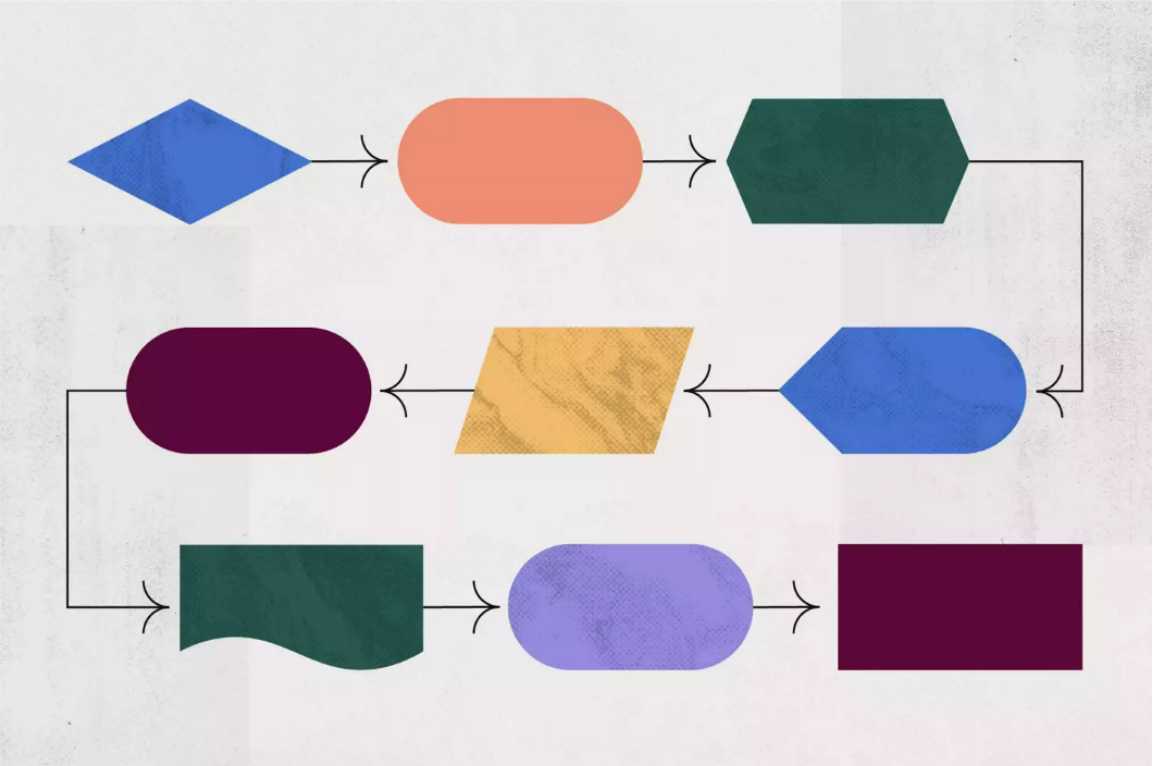
Fluxograma



Fluxograma é uma ilustração de fluxo de um processo através de etapas. Dentre suas utilidades, estão:

- Documentar um processo
- Simplificar e visualizar ideias ou processos complexos
- Organizar a equipe e atribuir tarefas de forma eficaz
- Tomar decisões e justificá-las
- Entre outras

Existem dezenas de símbolos para se criar um fluxograma, os mais comuns são:



Linguagens de programação

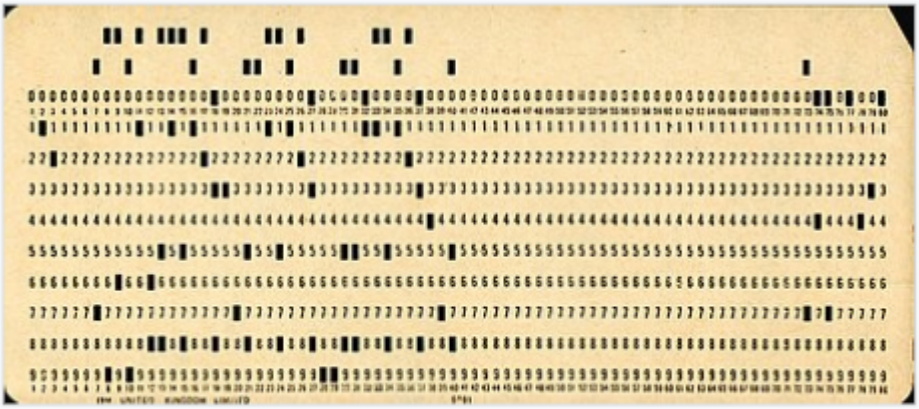
Existem dezenas de linguagens de programação (e todos os anos surgem mais). Neste início, é importante compreender que não existe uma linguagem melhor que a outra, mas sim linguagens **mais adequadas** para uma determinada situação. Quando falamos de linguagem de programação para sistemas embarcados, onde o foco é o máximo desempenho e comunicação quase direta com o hardware, a melhor linguagem, com certeza, é C.

A função da linguagem de programação é de facilitar ao máximo a comunicação entre ser humano e máquina. Apesar dos computadores conseguirem literalmente conversar com seres humanos hoje em dia, eles não entendem nada além de pulsos elétricos, ou bits.

Os bits são que de fato trafegam nos circuitos eletrônicos digitais, entre os componentes. Um microcontrolador só consegue receber um conjunto de bits e retornar outro conjunto de bits. O compilador faz o papel de traduzir tudo o que o programador escreve no código em bits.

Ouve um momento, no início da ciência da computação e da era de máquinas dirigidas por instruções, em que a forma mais comum de se programar um computador era por meio de cartões perfurados. Na confecção destes cartões, era

necessário programar o comportamento de cada bit (tanto seu estado quanto o sua função no hardware), e se houvesse algum erro, o cartão era perdido. Além disso, a leitura destes cartões pela máquina não era eficiente, podendo ficar emperrados ou algum furo passar sem ser lido, comprometendo todo o processo de dados.



Um cartão perfurado IBM de 12 linhas/80 colunas de meados do século XX

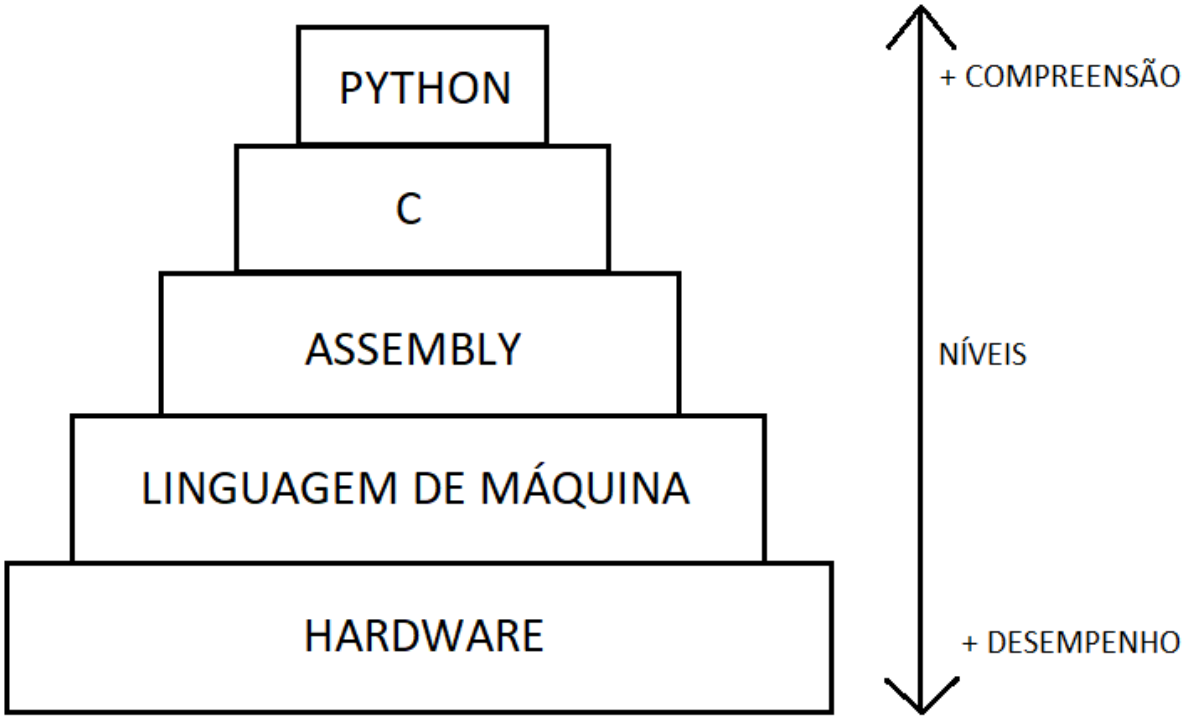
Para tornar o processo de programação mais eficiente e mais legível para humanos, surgiu o Assembly. Programar nesta linguagem pode ser feito diretamente pelo computador e, ainda hoje, ela pode ser utilizada para a programação de microcontroladores e microprocessadores. Sua grande vantagem é o controle total sobre o hardware em termos de consumo de memória e gerenciamento de threads, mas a desvantagem é que seu código assembly será compatível somente com máquinas idênticas as quais ele foi produzido. Veja um exemplo de um código em Assembly:

```
section .data
msg     db      'Como programar em Assembly - Curso Assembly Progressivo', 0AH
len     equ     $-msg

section .text
global  _start
_start: mov     edx, len
        mov     ecx, msg
        mov     ebx, 1
        mov     eax, 4
        int     80h

        mov     ebx, 0
        mov     eax, 1
        int     80h
```

Mas Assembly ainda é extremamente complexa e torna o processo de programar lento, principalmente para programadores iniciantes. Felizmente, devido aos esforços de muitos programadores na década de 1970, surgiu o C, uma linguagem

muito mais simples de entender, mais portável entre máquinas diferentes e ainda mantendo o bom controle sobre o hardware que o Assembly dá. Hoje, C é uma das linguagens mais utilizadas no mundo e serve como base de conhecimento para todo o programador profissional.

Enquanto o Assembly é considerada uma linguagem de baixo nível, ou seja, mais próximo da máquina do que do homem, C é uma linguagem de médio para alto nível, cuja sintaxe apresenta uma forma mais “humanizada” de se executar comandos, quase como se você estivesse dando instruções para uma pessoa real.



A linguagem C é utilizada em:

- Sistemas operacionais como Windows e Linux;
- Sistemas desktop e jogos;
- Sistemas embarcados, IOT, robótica, sistemas automotivos, sistemas de som.

C foca em desempenho e proximidade aos recursos

Compiladores x interpretadores

No universo das linguagens de programação, existem dois tipos de linguagens: as compiladas, que necessitam de um compilador instalado em sua máquina; e as interpretadas, que necessitam de um interpretador. A diferença entre um interpretador e um compilador é que o interpretador executa o código linha por linha,

já o compilador lê o programa inteiro e converte o mesmo em um código-objeto, algo que a máquina consegue entender e executar.

De forma prática, se o código é interpretado, qualquer mudança que você fizer no código vai refletir na execução do programa. Já em um código compilado a mudança não altera o programa, é necessário recompilar um novo programa para que as mudanças sejam aplicadas.

C é uma linguagem compilada. O editor de texto onde você escreve o código C não fará absolutamente nada além de te auxiliar nas questões de ortografia, sugestão de palavras e alguns atalhos. No momento que você pressiona compilar, o seu editor vai buscar o compilador instalado em sua máquina para realizar a conversão de seu código em um arquivo executável.

Funcionamento do compilador

Um compilador é um programa de software que traduz código fonte escrito em uma linguagem de programação de alto nível (como C, C++, Java, Python) em código de máquina ou em código intermediário executável por um computador. O processo de compilação envolve várias etapas, que geralmente incluem:

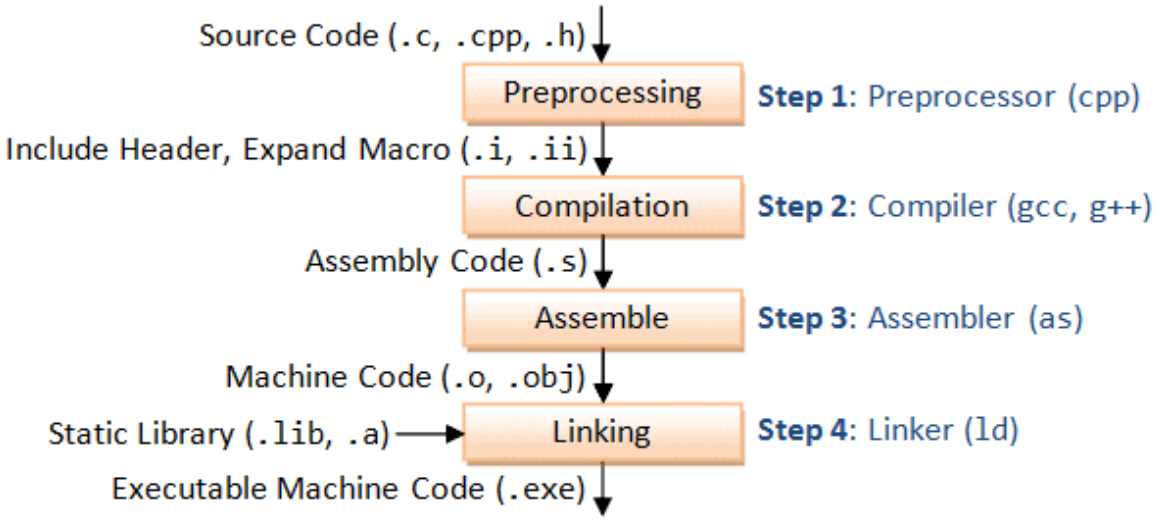
1. Análise léxica: O compilador lê o código fonte caractere por caractere e divide-o em tokens, identificando palavras-chave, identificadores, números, operadores, símbolos, etc. Esta fase também elimina comentários e espaços em branco desnecessários.
2. Análise sintática: O compilador verifica se a estrutura do código segue as regras da linguagem de programação em questão. Isso envolve a construção de uma árvore sintática ou uma tabela de símbolos que representa a estrutura hierárquica do código.
3. Análise semântica: Nesta fase, o compilador verifica se o código faz sentido em termos de tipos de dados, escopo de variáveis e outras regras semânticas da linguagem. Erros semânticos, como o uso de uma variável não declarada, são identificados aqui.
4. Geração de código intermediário: O compilador pode criar código intermediário que é uma representação de nível mais baixo e independente da máquina do código fonte. Isso facilita a otimização e a geração de código de máquina para diferentes arquiteturas de computador.
5. Otimização de código: Muitos compiladores incluem otimizações para melhorar a eficiência do código gerado. Isso pode envolver a reordenação de instruções,

a eliminação de código redundante e a substituição de operações complexas por versões mais eficientes.

6. Geração de código de máquina: Finalmente, o compilador traduz o código intermediário em código de máquina específico para a plataforma de destino. Isso pode envolver a alocação de registros, a resolução de endereços e outras tarefas específicas da arquitetura.
7. Ligação (se necessário): Em muitos casos, os programas são compostos por vários arquivos de código fonte. O linker (ligador) é responsável por combinar esses arquivos em um único programa executável. Isso pode incluir a resolução de referências entre os diferentes módulos.
8. Geração de executável: O resultado final é um programa executável que pode ser executado em um computador.

É importante notar que diferentes compiladores podem operar de maneira ligeiramente diferente e podem incluir etapas adicionais ou omitir algumas etapas, dependendo da linguagem de programação e das otimizações desejadas. Além disso, algumas linguagens de programação, como Python, usam uma abordagem diferente, interpretando o código fonte em tempo de execução em vez de compilá-lo para código de máquina.

Em resumo, um compilador é uma ferramenta fundamental no desenvolvimento de software, pois permite que os programadores escrevam código em linguagens de alto nível compreensíveis para os humanos e, em seguida, o traduzam em código de máquina que pode ser executado por um computador. Isso facilita muito o processo de desenvolvimento de software e permite que os programas sejam executados em diferentes plataformas de hardware.



Exemplo de algoritmo

```
Algoritmo Recepcionista de Cinema

Inicio
Solicitar ao cliente entrada para o filme.
Conferir o horário do filme no entrada.
    Se hora atual > hora do filme + 30 minutos Então
        Informar ao cliente que o tempo limite para entrada foi excedido.
        Não permitir a entrada.
    Senão Se hora atual < hora do filme - 30 minutos Então
        Informar ao cliente que a sala do filme ainda não foi liberada para entrada.
        Não permitir a entrada.
    Senão
    Permitir a entrada.
    Indicar ao cliente onde fica a sala do filme.
Fim
```