LINGUAGEM C

1. **ESTRUTURA BÁSICA DE UM PROGRAMA EM C:**

#include <>

int main () {

\\código a ser escrito

return 0;

}

1. #include: é uma diretiva do pré-processador. Ela inclui o conteúdo de um arquivo antes de copilar.
2. <>: biblioteca na linguagem C – Cada biblioteca no C serve para habilitar e/ou desabilitar certas funções. Exmplos de algumas bibliotecas e suas funções:

* <stdio.h> - essa biblioteca cuida da entrada e saída padrão do programa;
* <locale.h> - essa biblioteca serve para configurar regras culturais e de idioma no programa;
* <math.h> - essa biblioteca possui funções matemática.

1. int main () – define a funçõ principal, ponto de entrada do programa.

* int – indica que a função retorna um número inteiro (por convenção, 0 (zero) significa que deu certo);
* main () – nome da função principal.

1. { - ínicio do bloco do programa.
2. return 0 – finaliza a função “main”. Retorna o valor “0” ao sistema operacional, programa terminou sem erros.
3. ; - separador de instrução;
4. } – fim do bloco do programa.

* Sempre termina alguma função com ponto-virgula (“;”).

1. **ESTRUTURA COM BIBLIOTECAS**
2. <stdio.h> - StanDard Input/Output Header

Essa biblioteca cuida da entrada e saída padrão do programas

Exemplo de programa:

#include <stdio.h>

int main() {

char letra, nome[50], completo[50];

int idade;

float altura;

printf("Primeira letra do nome: ");

scanf(" %c", &letra);

printf("Primeiro nome: ");

scanf(" %49s", nome);

printf("Nome completo: ");

scanf(" %49s", completo);

printf("Sua idade: ");

scanf("%d", &idade);

printf("Sua altura: ");

scanf("%f", &altura);

printf("Primeira Letra: %c;\n", letra);

printf("Primeiro Nome: %s;\n", nome);

printf("Nome Completo: %s;\n", completo);

printf("Idade: %d;\n", idade);

printf("Altura: %.2f.", altura);

return 0;

}

### **Variaveis**

* Tipo nome – Ex:. char letra
* char – guarda caracteres – char letra = “a”, “G”, “6”;
  + char[x] – guarda uma certa quantidade de caractere + “0/” – char[5] = “abcd” + “0/”.
* int – números inteiros – ex.: 24, -9, 2025;
* float – números decimais – ex.: 3.14,1.75, -0.5;
* double – número muito grande – ex.: 300000.365893

Ex.:

Tipos de char a ser usado:

* char letra – lê apenas um caractere: “A”, “d”, “3”;
* char nome[50] – é uma *array* que vira uma *string* (texto), pois guarda a quantidade de caractere fornecido (50) menos 1 caractere, pois o ultimo caractere é “\0”.

**Funções**

1. Saída de dados (mostrar informações na tela):

* printf() – imprime texto formatado na tela;
* puts – imprime uma string e adiciona automaticamente um `\n` (quebra de linha);
* fputs – imprime uma string sem adicionar `\n` (mais usado em arquivos, mas também pode escrever na tela);
* putchar – imprime apenas um caractere,

1. Entrada de dados (ler informações do usuário):

* scanf() – usado para ler dados digitados pelo usuario. Ex.: scanf("%d", &idade);
  + %d – indica o tipo esperado (inteiro);
  + &idade – o operador “&” passa o endereço da variável, pois o “scanf” precisa saber onde guarda o valor digitado;
* fgets() – lê uma linha inteira de texto, incluindo espaços. Ex.: “fgets(nome, sizeof(nome), stdin);” - lê uma linha inteira de texto, incluindo espaços;
* getchar() – lê um caractere do teclado;
* gets() – NÃO USE (inseguro, pode causar falhas de segurança).

1. Leitura de dados em arquivos:

* fscanf() – lê dados formatados de um arquivo;
* fgets() – lê uma linha de texto de um arquivo;
* fread() – lê blocos de dados de um arquivo.

1. Escritas de dados em arquivos:

* fprintf() – escreve dados formatados em um arquivo;
* fputs() – escreve uma string em um arquivo;
* fwrite() – escreve blocos de dados em um arquivo.

1. Manipulação de arquivos:

* fopen() – abre um arquivo;
* fclose() – fecha um arquivo.

### **Especificadores**

* %c – caractere unico: char letra;
* %s – string, lê uma cadeia de caractere
  + Ao se ler a variavel char, deve inserir a quantidade de caractere que deve ser lido: %49s = “ProfessorWhoEstaVivo”;
  + Cuidado: as vezes ao digitar %s no “scanf”, a liguagem C interpreta o espaço entre as letras como “ENTER” e pode ocorrer erro e continuidade no programa – revisar para trocar por outra função ou escrever sem espaço.
* %d – imprime valor inteiro: int idade = 30;
* %i – equivalente a %d;
* %e – imprime um nuúmero de ponto flutuante na notação cientifica;
* %f – número decimal: float altura = 1.94
  + %.xf – pula a quantidade fornecida de casa depois do ponto: %.3f = 3.654.

**Sequência De Escape**

Em C, dentro de strings ("..."), existem símbolos especiais que começam com \.

* \n – significa *newline* (quebra de linha);
* \t – tabulação (tab);
* \\ – imprime uma barra invertida \;
* \" – imprime aspas ";
* \0 – caractere nulo (marca o fim de uma string).
* Um detalhe importante no uso do “scanf()” é que ele **não consome o ENTER (“\n”) digitado ao final da entrada.**

Esse ENTER fica guardado no **buffer de entrada** (a fila de caracteres que o teclado enviou).

Quando o programa chega no próximo scanf(), ele encontra esse ENTER que ainda está lá e interpreta como se você já tivesse digitado algo.

O resultado é que o programa "**pula**" a entrada e vai direto para a próxima instrução, o que pode confundir quem está aprendendo.

Formas de resolver:

1. Colocar um espaço antes do especificador no scanf(): scanf(" %49s", nome); - esse espaço faz o scanf() ignorar os ENTERs e espaços em branco anteriores;
2. Usar fgets() em vez de scanf() para strings: fgets(nome, 50, stdin) - o fgets() lê a linha inteira, incluindo espaços no meio do texto;
3. Consumir o ENTER manualmente com getchar(): getchar(); - essa função lê apenas um caractere (nesse caso, o ENTER que estava sobrando);
4. Usar o especificador %[^\n] no scanf() : scanf(“ %[^\n]”), nome); - assim o scanf() lê todos os caracteres digitados até encontrar um ENTER, aceitando inclusive espaços;
5. Limpar o buffer manualmente: while (getchar() != '\n'); - esse laço descarta o ENTER (e qualquer outro caractere que tenha sobrado) antes de continuar.
6. Expressões Regulares (Regex) - Não é uma função do C padrão — é um conceito usado em várias linguagens (C, C++, Python, Java, etc). São padrões de busca que permitem validar ou procurar texto de forma avançada. Em C, para usar expressões regulares, você precisa incluir bibliotecas externas, como **<regex.h>** no POSIX (Linux, Unix).

Exemplos de regex:

* ^[0-9]+$ - só aceita números.
* ^[A-Za-z ]+$ - aceita apenas letras e espaços.
* .+ - qualquer coisa, desde que tenha pelo menos 1 caractere.

Resumindo: regex é poderoso para validar strings, mas não é usado só para ler — é usado principalmente para verificar se o que o usuário digitou segue uma regra.

Diferença prática

fgets - lê a entrada do usuário de forma segura;

Regex - verifica ou filtra a entrada depois que você já leu.

* Apenas no **array** do **char** do scanf(), que contem o especificar “%s”, que não precisa do operados “**&**” O scanf() sempre precisa de um endereço de memória (onde vai guardar o valor digitado). Para variáveis simples (int, float, char), você precisa usar & para passar o endereço. Para **arrays** de **char** (**strings** em C), o nome do array já é um **ponteiro** para o primeiro elemento. Então não precisa usar &. Por que isso acontece:

1. int idade; - idade guarda um número, mas scanf precisa do endereço, então você usa &idade.
2. char nome[50]; → nome é um **array**, e em C o nome de um **array** já representa o endereço do primeiro elemento (&nome[0]). Então scanf("%s", nome) já recebe o endereço certo.

* Essa biblioteca não entende acento e nem sabe de idiomas. Pra ela, “a” e “ã” são só números diferentes da tabela de caractere.

1. <locale.h> Localization Header

Essa biblioteca serve para configurar regras culturais e de idioma no programa:

* Formatação de números – vírgulas (,) ou ponto (.) como separador decimal;
* Nome de meses/dias da semana depende do idioma;
* Codificação de caracteres (por isso conseguimos usar UTF-8)

Funções:

* setlocale(LC\_ALL, “pt\_BR.UTF-8”); - essa função fala pro programa: “use as conveções gramaticais da lingua Portugues-Brasil.
* Essa biblioteca não sabe imprimir nada na tela. Ela só muda como outras funções se comportam.

# **OUTROS**