**AS MANHAS DE C**

**PRIMEIRO CONTATO:**

1. Cada projeto C, suporta um programa C
2. interessante criar um sistema de pastas para guarda os códigos fontes dos exercícios

#include <stdio.h>

Int main(){

Printf(“Hello World!”);

Return 0;

}

**ENTENDENDO UM POUCO DA PRIMEIRA SITNAXE:**

#include <stdio.h> // é uma biblioteca arquivos cabeçalhos, que trabalha com funções de entrada de dados e saídas, como o printf

Int main(){} // chamada de função na qual qualquer código entre chaves será executado.

Printf(“”) // Basicamente é uma função que escreve na tela uam saída padrão qualquer, colocada entre parênteses

Por de trás, é como se fosse char nome[numero de caracteres]

Um array de letras.

Return 0; Finaliza a função main(), porem é um conceito mais avançado.

Dentro da int main, tudo que está dentro de { } sera executado.

Toda instrução ira terminar com ;

int main(){printf("Hello World!");return 0;}, é tudo uma linguiça de bits no final é assim que o compilador enxerga o código, mas espaçamos para ser mais legível.

**QUEBRAS DE LINHA:**

Printf(“Algo!”)

Printf(“Legal”)

Sairia como: Algo!Legal, logo o programa não adiciona espaços sozinho, você precisa usar o comando \n dentro da string.

Printf(“Algo!\n”)

Printf(“Legal”)

Saira:

Algo!

Legal

Podes encontrar mais quebras de linhas, nos códigos fontes.

**COMENTARIOS:**

Podem ser usados para deixar o código mais legível, explicando as partes deles,

Como também pode ser usado para comentar códigos alternativos, logo se um não der certo, descometa o outro possível código

**Sintaxe:** // Algo

Qualquer texto em uma linha depois de // é ignorado pelo compilador.

**Comentando varias linhas:**

/\* Pega tudo que está

Aqui dentro disso \*/

**VARIAVEIS:**

Em C existem diferentes tipos de variáveis, definidas por palavras chaves diferentes.

**Exemplo:**

Int – Armazena números inteiro, sem decimais, como 123 ou -123

Float – Armazena números de ponto flutuante, com decimais como 19,99 ou -19,99

Char – armazena caracteres únicos, como ‘a’ ou ‘B’, todo caractere estará entre aspas simples.

**DECLARANDO ESSAS VARIÁVEIS:**

**Exemplo de sintaxe:**

Tipo nomeVariavel = valor;

Ou

Tipo nomeVariavel;

**Exemplo real:**

Int meuNumero = 15;

Ou

Int meuNumero;

meuNumero = 15; //exemplo de atribuição do valor.

Printf(“%n”, meuNumero)

// já já falaremos sobre as formatações de saídas de dados. Ou melhor “Especificadores de formato”

OBS: Se você atribuir um valor a uma variável existente, ela substituirá o valer anterior.

**ESPECIFICADORES DE FORMATO:**

São usados juntos ao Printf()

Que será basicamente um espaço reservado para informar ao compilador o tipo de valor da variável que sera gerado dentro da função Printf.

**Especificadores em si:**

Int – “%d” ou “%i” //para valores inteiros.

Char – “%c”

Float – “%f”

**Exemplo:**

**#include <stdio.h>**

int main(){

// declarando variaveis

int inteiro = 5;

float real = 31.55;

char nome = 'D';

// Printando valores

printf("%d\n", inteiro);

printf("%0.1f\n", real); // %0.2f" o 0.2 define quantas casa decimais o numero exibira.

printf("%c\n", nome); // só recebe um caractere, pois para declarar nomes inteiros, é necessário fazer um array de char.

return 0;

}

**DECLARANDO UMA VARIÁVEL COM UM TEXTO:**

Int meuNumero = 5;

Printf(“Meu numero favorito é: %d”, meuNumero)

Ou imprimindo diferentes tipos de números:

Int numb = 5;

Char letra = ‘n’;

Printf(“Meu numero e: %d e minha letra é %c”, numb, letra);

**Adicionando variáveis juntas:**

Int x = 5;

Int y = 6;

Int sum = x + y;

Printf(“%d”, sum);

**Declarando várias variáveis:**

Int x = 5, y = 6, z = 50;

Printf(“%d”, x + y + z);

**Nomes:**

Obviamente é indicado que, usem-se nomes descritivos no padrão camel case, já conhecido.

1. Nome podem conter letras, dígitos e sublinhados
2. Nomes devem começar com uma letra ou sublinado(\_)
3. Os nomes diferenciam maiúsculas e minúsculas! Myvar e myvar são variáveis diferentes para o compilador.
4. Os nomes não podem contem espaços em brancos ou caracteres especiais, (#,!,% etc)
5. Palavras reservadas não podem ser usadas! Int, char, float etc.

**TIPOS DE DADOS E SEUS ESPECIFICADORES:**

**Int** – 2 ou 4 bits , armazena números inteiros **(“%d”** ou **“%i” )**

**Float** – 4 bits , armazena números fracionados, contendo um ou mais decimais e é suficiente para armazenar 7 dígitos. **(“%f”)**

**Double** – 8 bits, armazena números fracionários, contendo uma ou mais casas decimais. É suficiente para armazenar um número de 15 dígitos decimais.

**(“%lf”)**

**Char** – Armazena um único caractere/letra/número ou valores da tabela ASCII.

**(“%s”)** usado para strings de textos, nas quais veremos em breve

Se for so para exibir uma letra pode usar o “%c”

**(“%lu”)** Usado para exibir o tamanho em bytes de uma variável.

**(“%P”)** Exibição de ponteiros.

.

**CONSTANTES:**

São usadas para a criação de variáveis constantes que não são mutáveis de forma alguma

Pra isso, usamos a palavra const

Assim a variável será imutável, somente servindo para leitura.

Exemplos:

Const int myNum = 15;

myNum = 10; // Vai acontecer um erro, logo a variável myNum é umas constante imutável.

Sempre devemos declarar constantes, em casos de valores que obviamente não serão alterados

Exemplo:

# include <stdio.h>

int main()

{

const int MINUTESPERHOUR = 60;

const float PI = 3.14;

printf("%d\n", MINUTESPERHOUR);

printf("%0.2f", PI);

return 0;

}

OBS: Ao criar uma varaivel constante, nela já se deve atribuir um valor, logo, isso não é possível:

Const int minutesPerHous;

MINUTESPERHOUR = 60;

Printf(“%d”,MINUTESPERHOUR); // isso daria erro, logo MINUTEPERHOUR esta com o valor constante vazio, e o mesmo não pode ser mudado.

É uma boa pratica, declarar variáveis constantes com letras maiúsculas.

**OPERADORES**

Usamos operadores para fazer operações em variáveis.

Já veremos também a ordem de resolução.

Exemplo:

Int myNum = 100 + 50;

Podemos somar também, variável com valor, duas variáveis, etc..

Int sum1 = 100 + 50;

Int sum2 = sum1 + 250;

Int sum3 = sum1 + sum2;

C divide os operadores nos seguintes grupos:

• Operadores aritméticos

• Operadores de atribuição

• Operadores de comparação

• Operadores lógicos

• Operadores bit a bit

**OPERADORES ARITIMETICOS**

*ADICAO:* ( + ) exemplo padrão!!

Int x = 5;

Int y = 3;

Printf(“%d”, x + y)

//Ou

Int z = x + y

Printf(“%d”, z)

//Ou

Int z;

z = x + y

printf(“%d”, z)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*SUBTRACAO:* ( - )

Int x = 5;

Int y = y;

Printf(“%d”, x - y)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*MULTIPLICACAO:* ( \* )

Int x = 5;

Int y = y;

Printf(“%d”, x \* y)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*DIVISÃO:* ( / )

Int x = 5;

Int y = y;

Printf(“%d”, x / y)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*MODULO:* ( % ) resto da divisão

Int x = 5;

Int y = 2;

Printf(“%d”, x % y); // que é igual a: 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*INCREMENTACAO:* ( + + ) Adiciona mais 1

Int x = 5;

Printf(“%d”, ++x) // vai da 6

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*DECREMENTACAO:* ( -- ) Diminui 1

Int x = 5;

Printf(“%d”, --x) // vai dar 4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Podem ser executados tanto dentro de uma variável, como dentro do print

OBS:

A seguir, vamos ver que é necessário dividir as linhagens de operadores, e uma explicação mais detalhada sobre os operados bit a bit, logo são para coisa de muito baixo nível, que vamos intender já já.

Importante saber que a própria Microsoft, tem algumas dicas peculiares sobre C e sua biblioteca CRT.

**OPERADORES DE ATRIBUICAO**

Para entender alguns operadores de atribuição é necessário entender programação bit a bit, porem vamos lá.

=

Atribuímos alguma coisa ou valor a variável.

Int = 5;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

+=

Atribui a soma de um valor a uma variável que já tem um valor.

Int x = 5;

X += 3; // resultado 8 5 + 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-=

Atribui uma diminuição ao valor de uma variável que já tem um valor

Int = 5;

x -= 3; // resultado 2 5 – 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*=

Atribui uma multiplicação a variável

Int x = 5;

X \*= 3; // resultado 15 5 \* 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/=

Atribui um divisão a variável

Float x = 5;

X /= 3; // 1.6667 5 / 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

%=

Atribui ao resto da divisão de uma variável

int x = 5;

X %= 3; // 2 5 % 2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

&=

Atribuição AND bit a bit

Int x = 5;

X &= 3; // 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|=

Atribuição OR inclusivo bit a bit

Int x = 5;

X |= 3; // 7

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

^=

Atribuição OR exclusivo bit a bit

Int x = 5;

X ^= 3; // 6

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>>=

Atribuição shift direita

Int x = 5;

X >>= 3; // 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<<=

Atribuição shift esquerda

**OPERADORES DE COMPARACAO**

Esses operadores obviamente, são usados para comparar dois valores.

O valor de retorno de uma comparação pode ser:

Tru (1)

False(0)

Exemplo rápido:

Int x = 5 ;

Int y = 3;

Printf(%d, x > y); // retar 1, pois 5 é maior que 3

==

Igual

X == y

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

!=

Diferente

X != y

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>

Maior que

X > y

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<

Menor que

X < y

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

>=

Maior ou igual a

X >= y

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<=

Menor ou igual a

X <= y

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**OPERADORES LOGICOS**

Esses operadores são usados para determinar a lógica das variáveis

Exemplo:

Int x = 5;

Int y = 3;

Printf (“%d”, x > 3 && x < 10) // retora 1, pois 5 é maior que 3 e menor que 10, já vimos tabelas verdades antes.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

&&

AND

Retorna verdadeiro somente se as duas comparações forem verdadeiras.

X < 5 && x < 10

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

||

OR

Retorna verdadeiro se uma das duas comparações forem verdadeiras

X < 5 || x < 4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

!

NOT  
Nega uma comparação total, ou seja, se algo sai de um valor verdadeiro de uma expressão comparativa, ele vai virar negativo, e verse versa

!(x < 5 && x < 10)

**TAMANHO DO OPERADOR**

Podemos ver o tamanho da memoria em Bytes de um tipo de dado ou variável usando:

Sizeof com a especificação %lu

Exemplo:

Int inteiro;

Float flutuante;

Double real;

Char carac;

Printf(“%lu\n”, sizeof(Inteiro));

Printf(“%lu\n”,sizeof(flutuante));

Printf(“%lu\n”,sizeof(real));

Printf(“%lu\n”,sizeof(carac));

// 4,4,8,1

**OPERADOR DE INCREMENTO**

++ incrementa +1

-- decrementa -1

X++ Pos incremento

++X pre incremento

X-- Pos decremento

--X Pre decremento

int main()

{

    /\*

1. soma -> +=

        2) subtração -> -=

        3) multiplicação -> \*=

        4) divisão -> /=

        5) resto (módulo) -> %=

    \*/

    int i = 1;

    printf( “%i\n”, i );

    printf( “%i\n”, ++i );

    i++;

    printf( “%i\n”, i );

    system(“cls”);

    int i2 = 5;

    printf(“%i”, i2);

    printf(“%i”, --i2);

    i2--;

    printf(“%i”, i2 );

    /\*

1. Incremento\

        >Pré ou Pos

        2) Decremento/

        Pre -> o valor será incrementado/decrementado na instrução

                que a variável estiver contida

        Pos -> o valor será incrementado/decrementado na próxima

                instrução

    \*/

    int x = 0;

    x = x + 10;//incrementar qntas unidades desejarmos

    x += 10;//incrementar qntas unidades desejarmos

    ++x;// op. Incremento, só podemos incrementar

        // uma única unidade

    system( “cls “);

    printf(“%i”, x);

    return 0;

}

**IF , ELSE**

Para fazer estruturas condicionais usaremos:

If, else, else if

E depois veremos o Switch

**Exemplo de sintaxe:**

if (Condição)

{

(execução caso seja verdade);

}

// exemplo real

if (20 > 18)

{

Printf(“Ual em man.”);

}

**Obviamente também podemos testar variáveis.**

Int x = 4;

Int y = 7;

if x > y

{

Printf(“Muito baum”);

}

**Agora a sintaxe com else**

if (úmeros)

{

(bloco de comando);

} else

{

(bloco de comando);

}

**Exemplo:**

Const int time = 20;

If (time < 18)

{

Printf(“Bom dia”)

}

Else

{

Printf(“Boa noite”)

}

**Sintaxe else if**

If (condicao)

{

(bloco de úmeros);

}

Else if (condicao2)

{

(Bloco de úmeros);

}

Else

{

(Bloco de úmeros);

}

**Exemplo:**

Const int time = 22;

if (time < 10)

{

Printf(“Boa manhã”);

}

else if (time < 20)

{

Printf(“Bom dia”);

}

Else

{

Printf(“Boa noite”);

}

Por isso é bom você já ter feito o curso de logica em visualg que vai te dar as primeiras ideais para resolver certas coisas simples.

**Forma curta de se fazer um se**

Int time = 20;

(time < 18)? Prinf(“Bom dia”) : printf(“Boa tarde”);

É uma abreviação de

Int time = 20;

if (time < 18)

{

Printf(“Bom dia”);

}

Else

{

Printf(“Boa tarde”);

}

**SWITCH**

Interruptores, são uma forma mais fácil, para não fazer muitos ifs e elses, usados para quando você sabe com certeza os cenários possíveis.

**Exemplo:**

Int day = 4;

Switch (day)

{

Case 1:

Printf(“Monday”);

Break;

Case 2:

Printf(“Tuesday”);

Break;

Case 3:

Printf(“Wednesday”);

Break;

}

Ou no último case, pode usar default para sinalizar um bloco a ser executado caso a variável não atenda nenhum dos casos.

Int day = 4;

Switch (day)

{

Case 1:

Printf(“Monday”);

Break;

Case 2:

Printf(“Tuesday”);

Break;

Case 3:

Printf(“Wednesday”);

Break;

Default:

Printf(“Não intende!”);

}

**LOOPINGS**

**WHILE:**   
Percorre um bloco de código desde que a condição especifica seja true.

Ex:

Int i = 0

While (i < 5)

{

Printf(“ %d\n”,int );

i++;

}

**DO WHILE**

Executa o looping ou um código completo,uma vez, antes de entrar no while em ci e testar uma condição para iniciar o looping.

Ex:

Int = 0;

Do

{

Printf(“%d\n”, i);

I++;

}

While(1 < 5);

“Faz declaração, enquanto a resposta para a pergunta for verdadeira.”

Fazer

Saída “Qual é a sua idade?”

Insira a idade\_do usuário

Saída “Qual é a idade do seu amigo?”

Insira amigo\_idade

Saída “Juntas suas idades somam: “

Saída age\_user + friend\_age

Saída “Deseja tentar novamente? S ou n “

Loop\_response de entrada

Enquanto loop\_response == ‘y’

**FOR**

Usado para quando sabemos exatamente quantas vezes queremos percorrer o looping.

Dentro do For:

Statement 1: Executa o código uma vez antes da execução do bloco, logo determina o valor inicial que o i valera

Statement 2: Define a condição para executar o bloco de código de quantas vezes o algoritmo percorre o looping

Statement 3: Executa o código todas as vezes após a execução do bloco de códigos ou incrementa um numero amais no i

Ex: Imprime 0 a 4:

Int i;

For (i = 0; i < 5, i++)

{

Printf(“%d\n”, i);

}

// Podemos brincar com as expressões do looping de acordo com a necessidade

For (I = 2; i+1 < 6; i++) ou

for (i = 1; i-1 < 6; i++)

que imprimiria de 1 até 6

ou

// imprime úmeros pares.

Int i;

for(i = 0; i <= 10; i = i + 2)

{

    printf(“%d\n”,i);

}

 return 0;

}

**BREAK AND CONTINUE**

**Break:**

Vimos que no switch o break salta um bloco de comando caso a condição verdadeira

Em loopings, break pode ser usado para sair de um looping, quando por exemplo alguma condição for verdadeira.

Int main()

{

// executa o looping enquanto i ate que i seja igual a 4

    int I;

    for(I = 0; I < 10; i++)

    {

        if (I == 4)

        {

            break;

        }

        printf(“%d\n”, i);

    }

return 0;

}

**Continue:**

Interrompe uma ação do looping, caso ocorra uma condição especifica

E continua com a próxima interação do looping

// imprime de 0 a 10 pulando o numero 4

    int I;

 for(I = 0; I < 10; i++)

 {

   if (I == 4)

   {

     continue;

   }

   printf(“%d\n”,i);

 }

return 0;

}

**Podemos usar ambos loops While:**

// imprime de 0 ate 3

int main()

{

int I = 0;

while(I < 10)

{

    if(I == 4)

    {

      break;

    }

   printf(“%d\n”, i);

   i++;

}

return 0;

}

// imprime de 0 a 9 pulando o 4

int main()

{

int I = 0;

while(i < 10)

{

    if(i == 4)

    {

      i++;

      continue;

    }

   printf(“%d\n”, i);

   i++;

}

return 0;

}

**MATRIZES / ARRAYS**

Usado para armazenar vários valores em uma única variável, invés de declarar varias separas.

Um array só pode ter um tipo de dado.

//sintaxe array

    int Algo[100] = {}; // O numero de elementos sempre precisa ser

    // pre calculado, nesse caso as pocisoes de 0 a 100 não tem nada

    int algo2[100] = {12, 31, 52, 51, 522}; // com elementos ja postos

**Acessando os elementos:**

Basta consultar o índice da matriz, como já vi no curso de logica.

// lembrando que 0 é a primeira posicao.

    int algo[] = {25, 50, 75, 100};

    printf("%d", algo[0]);

    return 0;

**Alterando um elemento da matriz:**

// lembrando que 0 é a primeira posicao.

    int algo[] = {25, 50, 75, 100};

    // Alterando um elemento

    algo[0] = 80;

    printf("%d", algo[0]);

    return 0;

    // saida sera 80

**Loop através de uma matriz:**

   //looping por matriz

   int teste[] = {25, 50, 75, 100};

   int i ;

   for (i = 0; i < 4; i++)

   {

        printf("Pocisão %i - valor - %i\n",i,teste[i]);

   }

   // printa todos os numeros da matriz

**STRINGS**

São majoritariamente usadas para armazenas textos de caracteres.

“Hello World” é uma string de caracteres.

C não possui um tipo “String” para criar facilemente uma variável de string. Invés disso, aqui usamos um array de char, para fazer um texto.

É exatamente isso que a função prinft faz.

Exemplo:

 char texto[] = "Hello World!";

    printf("%s\n",texto); // imprime o texto

    printf("%c", texto[2]); //chama a letra "l"

**Modificando uma string:**

    char name[] = {"Hello World"};

    name[2] = 'Z';

    name[3] = 'z';

    printf("%s", name);

// HeZzo World

**Outra maneira de criar strings:**

    char name[] = {'H', 'e', 'l', 'l', 'o', ' ', 'J', 'e', 'i', 'e', 'l', '!', '\0' };

    printf("%s", name);

    return 0;

\0 é um caractere de terminação nulo.

Diz ao compilador que esse é o fim da string.

Ambos têm os mesmos 13 caracteres. Logo darão 13 bytes

   char name[] = {'H', 'e', 'l', 'l', 'o', ' ', 'J', 'e', 'i', 'e', 'l', '!', '\0' };

    char name2[] = {"Hello Jeiel!"};

    printf("%lu - Bytes\n", sizeof(name));

    printf("%lu - Bytes", sizeof(name2));

     // 13 bytes os dois

    return 0;

**Caracteres Especiais:**

As vezes escrever os textos entre aspas das strings, pode gerar um erro

A solução seria o caractere de escape de barra invertida.

Exemplo:

Char txt[ ] = {“Eles eram chamados de “vinkings” no norte europeu”};

Jeito certo

Char txt[ ] = {eles eram chamados de \” Vikings\” no norte europeu}

Outro exemplo

Char txt[ ] = {“It\’ alright.”};

**Caracteres de escape em ci:**

\’ , \” , \\

Sempre é a barra primeiro e o caractere depois.

\n new line

\t tab

\0 null

**COMANDOS DE ENTRADA DO USUARIO**

Usamos: Scanf() pra ler valores digitados

    int numb;

    printf("Digite seu numero: ");

    scanf("%d", &numb,"\n");

    printf("Seu numer é: %d", numb);

    return 0;

O Scanf sempre recebe o tipo de valor que será digitado nele e o & antes da variável, que é indicando para armazenar em algum canto da memória.

**ENDERECO DE MEMORIA**

Ao se criar uma variável em C, um endereço de memória é atribuído a ela.

**O endereço** é o local aonde a variável esta armazenada no computador.

Ao atribuirmos um valor a variável, o mesmo é armazenado neste endereço de memória.

Podemos descobrir aonde um resultado está armazenado, ao pedir para o compilador imprimar a variável, porem com “&” antes.

 int numb;

    printf("Digite seu numero: ");

    scanf("%d", &numb,"\n");

    printf("Seu numer e: %d", numb,"\n");

    printf("%p", &numb);

Digite seu numero: 23

Seu número e: 23

0x0061FF1C

0x = valor hexadecimal

0b= valor binário

    int numb;

    printf("Digite seu numero: ");

    scanf("%d", &numb,"\n");

    printf("Seu numer e: %d\n", numb);

    printf("POSICAO: 0x%p", &numb);

É apenas uma gambiarra minha para exibir de forma mais nítida!

Digite seu número: 32

Seu número e: 32

POSICAO: 0x0061FF1C

**PONTEIROS**

&variável, são conhecidos como ponteiros

Ponteiros são variáveis que armazenam o endereço de memória de outras variáveis como seu valor.

Uma variável de ponteiros aponta para um tipo de dado (como int) do mesmo tipo e é criado com o operador: \*.

Assim o endereço da variável na qual você está trabalhando é atribuído ao ponteiro.

Vamos ver mais do raciocínio por imagens, e tem mais nas minhas anotações.

Exemplo:

  int idade = 32;

    int\* ponteiro = &idade; //ponteiro recebe a posicao de idade

    printf("%d\n", idade);

    printf("%p\n", &idade); // exibi a posicao original da variavel idade

    printf("%p\n", ponteiro); // exibi a posicao do ponteiro.

    printf("%d", \*ponteiro); // aponta o valor apontado

32

0061FF18

0061FF18

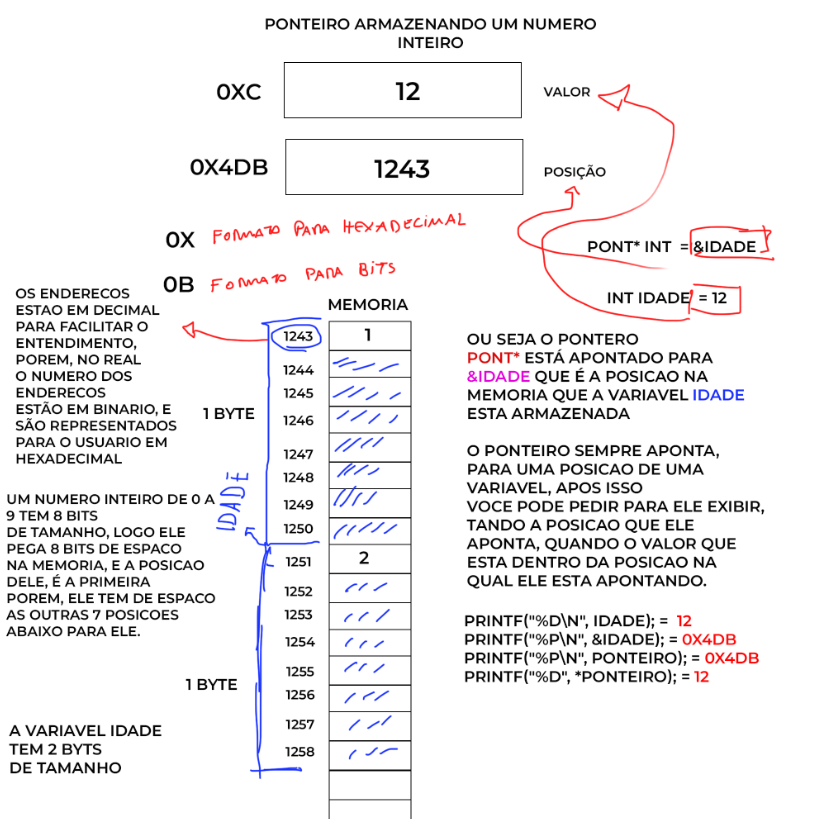
32

Enfim, ponteiros a longo ponto, é um assunto cabeludo, pois ajuda na manipulação de dados da memória, aumentando o desempenho de grandes programas

Logo, se forem mal manuseados, podem alterar o valor de dados armazenados em outros endereços.

Pois, vc já deve ter em mente, que é possível fazer mil coisas com a posição ou valor armazenando em um ponteiro.

Se aprofundando um pouco mais sobre isso:



**FUNCOES**

**Ainda falta ver operadores bit a bit**