**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

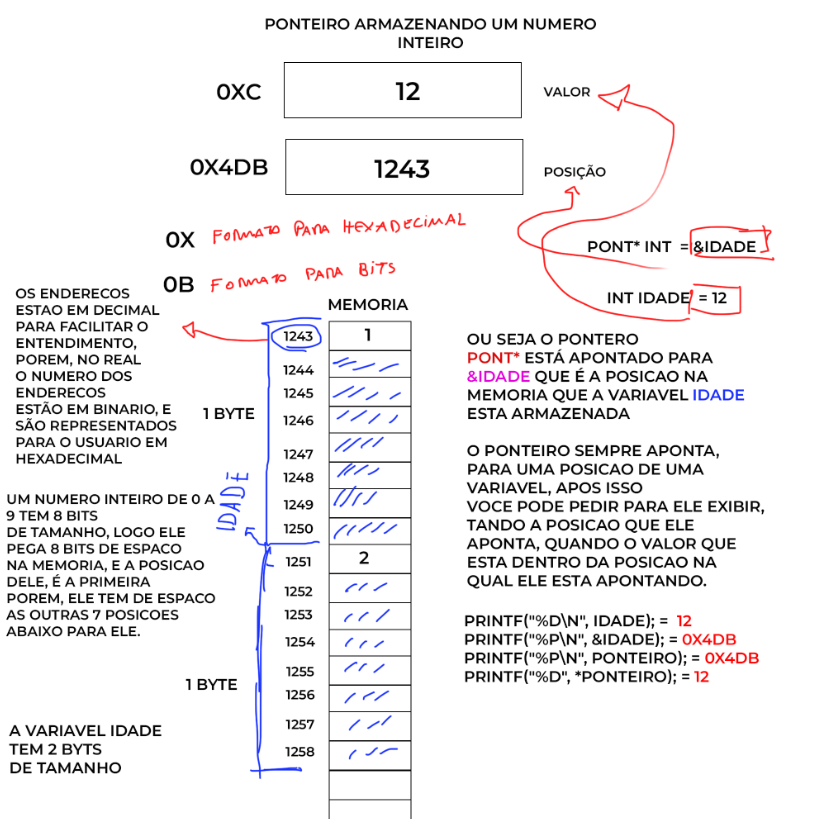
32

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**

É um bloco de codigo que so é executado quando chamado.

Voce pode passar dados, conhecidos comumente como pararmetros, para uma funcao, que são as tais variaveis de escopo.

É importante para a reutilizacao de codigo, e diminuacao do codigo principal, pois eu imagino que grandes projetos são compostos por bibliotecas propias que serão integradas no final ao progama main.

Esse tempo todos já estavamos usando funcoes, por exemplo

Int main(){} é uma funcao que executa os codifos dentro das chaves

E printf() é uma funcao usada para imprimir textos na tela.

**Criando uma funcao:**

  void myFunc()

  {

    printf("Hello Worlds!");

  }

int main()

{

  myFunc();            // executa a mensagem, que esta dentro da funcao.

  return 0;

}

A funcao sera executada, em ordem, quantas vezes for declarada.

Da para fazer, umzilhão de coisas com isso, vai depender da sua imaginação

void Message()

{

  printf("Isso e apenas um teste.\n");

}

int main()

{

int i;

  for (i = 0; i < 3 + 1; i++)

  {

     Message();

  }

  return 0;

}

**Passando parametros e argumentos para uma funcao:**

Informacoes podem ser passadas para uma funcao como parametros. Os parametros

Atuam como variaveis dentro da funcao(variaveis de escopo)

Os parametros são especificados dentro da funcao, entre parenteses.

Voce pode adicionar quantos parametros quiser, basta separalos com

Uma virgula.

Lembrando que tabem a como a funcao mudar globalmente

O valor de uma variavel passada como parametro.

Aqui vemos um exemplo passando um único parametro:

void nome(char name[])

{

  printf("Bem vindo: %s\n", name);

}

int main()

{

  char nome\_um[20];

  char nome\_dois[20];

  char nome\_tres[20];

  printf("Qual o nome da sua equipe? ");

  scanf("%s", &nome\_um);

  scanf("%s", &nome\_dois);

  scanf("%s", &nome\_tres);

  nome(nome\_um);         // eu estaria passando de qualquer forma "nome"

  nome(nome\_dois);

  nome(nome\_tres);

  return 0;

}

Aqui um exemplo passando varios parametros de uma vez:

void nome(char name[], int idade)

{

  printf("Bem vindo: %s  Sua idade e: %d\n", name, idade);

}

int main()

{

  char nome\_um[20];

  char nome\_dois[20];

  char nome\_tres[20];

  int idade\_um;

  int idade\_dois;

  int idade\_tres;

  printf("Qual o nome da sua equipe? ");

  scanf("%s", &nome\_um);

  scanf("%s", &nome\_dois);

  scanf("%s", &nome\_tres);

  printf("\n");

  printf("Qual a idade de ambos? ");

  scanf("%d", &idade\_um);

  scanf("%d", &idade\_dois);

  scanf("%d", &idade\_tres);

  nome(nome\_um, idade\_um);         // eu estaria passando de qualquer forma "nome" e um numero inteiro

  nome(nome\_dois, idade\_dois);

  nome(nome\_tres, idade\_tres);

  return 0;

}

Lembrando que grandes projetos, voce faz suas propias bibliotecas, e puxa suas funcoes de um arquivo de extensao .h

Essa biblioteca deve esta na pasta librarys do compilador, e deve esta executada na pasta de Binarios, do mesmo compilador, para ela já ser precarregada.

**Valores de retorno:**

A palavra “VOID” indica que a funcao em ci não deve retornar nenhum valor

Agora, se voce quiser que a funcao retorne um tipo de valor, voce pode usar, tipos de dados como: int, float, etc.. e usar a palavra return no fim da funcao.

Exemplos

#include <stdio.h>

int valor(int x)                     // retornando a soma de um valor e qualquer coisa

{

  return 5 + x;

}

int func2(int x, int y)           // retornando a soma de dois valores juntos etc.etc.etc a imaginacao e o limite.

{

  return x + y;

}

int somas(int t, int z, int j)

{

  return t + z + j;                          // passando o valor de uma funcao para uma varaivel:

}

int main()

{

  int numb = 32;

  int numb1 = 3;

  int numb2 = 5;

  int result = somas(numb1, numb2, numb);

  printf("O RESULTADO DISSO E: %d\n", valor(3));

  printf("OU O RESULTADO E: %d\n", valor(numb));

  printf("RETORNO DA SOMA DE DOIS VALORES: %d\n", func2(numb1, numb2));

  printf("MOSTRANDO A VARIAVEL QUE CONTEM O VALOR DA FUNCAO: %d", result);

  return 0;

}

**Declarando diversas de funcoes:**

No geral, funcoes cosistem em duas partes:

Declaracao: O nome da funcao, tipo de rretorno e parametro (se houver)

Definicao: O corpo da funcao (codigo a ser executado)

Para melhor otimizar o codigo e recomendado separar, a declareaca e a definicao da funcao.

Por mais estranho que pareça, frequentimente pode ser visto progamas em C que declarao a funcao antes de main,

E colocao as definicoes após ela. Pois é uma boa pratica

Porem isso é apenas quando voce já tem os valores da varaivel pre definidos, se não, não invente, pois ele vai pegar o valor aleatorios das variaveis que já existe, logo pela fato de C é uma linguagem que executa em ordem de cima para baixo.

int somas(int, int);

int main()

{

  int num1;

  int num2;

  int soma = somas(num1, num2);

  printf("Escreva um numero: \n");

  scanf("%d", &num1);

  printf("Escreva outro numero: \n");

  scanf("%d", &num2);

  printf("Resultado e: %d", soma);

  return 0;

}

int somas(int x, int y)

{

return x + y;

}

O resultado da soma vai dar um valor nada haver, pois so funcionaira se voce já jogasse direto o valor dos numeros na funcao tipo:

Int soma = Int somas(4, 6)

// boa pratica na declaracoa de funcoes

#include <stdio.h>

int somas(int x, int y)

{

return x + y;

}

int main()

{

  int num1;

  int num2;

  printf("Escreva um numero: \n");

  scanf("%d", &num1);

  printf("Escreva outro numero: \n");

  scanf("%d", &num2);

  printf("Resultado e: %d", somas(num1, num2));

  return 0;

}

Desse jeito da certo.

**Recursão:**

É uma tecnica de fazer chamadas de funcoes entre ci.

Essa tecnica acaba por oferecer uma maneira de quebrar problemas

Complicados em problemas simpels que são mais facies de resolver.

Exemplo: Adicionar dois numero é facil, mas adicionar um intervalor de numeros é mais complicado.

No exemplo asseguir, a recursão é utilizda para somar um intervalo de numeros, dividindo na simples tarefa de somar dois numeros:

Lembrando que é usado a bilbioteca Math.h

int sum(int k);

int main()

{

int result = sum(10);

printf("%d", result);

return 0;

}

int sum(int k)

{

  if (k > 0)

  {

    return k + sum( k - 1);

  }

  else

  {

    return 0;

  }

}

É basicamente (nesse caso) um looping da xuxa, quando a funcao sum() é chamada

ela adiciona o parametro K a soma de todos os numeros menores que K

e retorna o resultado. Quando K se torna igual a 0, a funcao retorna apeanas 0

ao executar o progama faz os seguintes passos.

O desenvolvedor deve ter cuidado ao escrever um recursão, pois assim como loopings, é muito facil escrever uma funcao que nunca termina ou uma que usa quantidade excessivas de memória ou poder de processamento.

No geral, recursões tem uma boa abordagem, matematicamente muito elegante.

Pode-se basicamente fazer uma boa sequencia de fibbonaci com isso, muito mais pratica de como é feito com o looping.

Essa é a tal tecnica que colocar que muda o valor de uma variavel após entrar dentro de uma funcao, tornando o valor Global e alterado.

int fibonacci(int i) {

if(i == 0) {

return 0;

}

if(i == 1) {

return 1;

}

return fibonacci(i-1) + fibonacci(i-2);

}

int main() {

int i;

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

printf("%d\t\n", fibonacci(i));

}

return 0;

}

Temos tambem exemplo de bilbioteca de funcoes que puxam:

floor(x) - Arredonda para Baixo

ceil(x) - Arredonda para cima

sqtr(x) - Calcula a Raiz quadrada

pow(x, y) - potenciacao

sin(x) - seno

cos(x) - cosseno

tan(x) - tangente

log(x) - logaritmo natural

log10(x) - logaritmo base 10

**STRUCTS**

É uma maneira de agrupar varias variaveis relacionadas em um só lugar, talves uma pensando de uma forma mais abstrata, seja tipo “Objetos”

Cada variavel na estrutura é conhecida como um **membro** da estrutura.

Diferente de um array, uma estrutura pode conter muitos tipos de dados diferentes (int, float, double, char, etc. )

**Criando uma estrutura:**

Usamos a plavra “sctruc” o nome e abre chaves para declarar seus menbros.

Exemplo:

Struct teste

{

Int meu\_numero;

Char Minha\_letra;

};

// deve-se sempre terminar uma estrutura de struct com ponto e virgula.

**Para acessar a estrutura, deve-se criar uma varaivel dela,**

Nesse caso, o nome dessa varia que vai armazenar o struct sera “s1”

Veja:

Struct teste

{

Int meu\_numero;

Char minha\_letra;

};

Int main()

{

Struct teste s1;

Return 0;

}

**Acessando os menbros de uma estrutura:**

Para isso, usaremos a sintaxe (.) ponto

Veja:

Struct teste

{

Int numero;

Char letra;

};

Int main()

{

Struct teste s1;

S1.numero = 12;

S1.letra = ‘c’;

Printf(“Minha letra é %c\n”, s1.letra);

Printf(“Meu numero é %d\n”, s1.numero);

Return 0;

}

Como notado, é possivel ir criando varias variaveis de estruturas com valores diferentes, usando apenas a mesma estrutura, so renomeando novas “variaveis de estrutura”

#include <stdio.h>

struct teste

{

int numero;

float flutu;

char letra;

};

int main()

{

struct teste s1;

struct teste s2;

s1.numero = 31;

s1.flutu = 31.441;

s1.letra = 'h';

s2.numero = 623;

s2.flutu = 3154.51;

s2.letra = 'd';

printf("Letra da estrutura 1: %c\n", s1.letra);

printf("Letra da estrutura 2: %c\n", s2.letra);

printf("Numero da estrutura 1: %d\n", s1.numero);

printf("Numero da estrutura 2: %d\n", s2.numero);

printf("Fluturante da estrutura 1: %0.2f\n", s1.flutu);

printf("Flutuante da estrutura 2: %0.2f\n", s2.flutu);

return 0;

}

**Ainda falta ver operadores bit a bit**

Progamas em C, no seu geral é composto por variaias bibliotecas, algo.h, que no fim são referenciadas no progama.c principal, aonde contem a int main.

Bibliotecas podem ser formadas por outras bibliotecas e etc, a imaginação é o limite.

Ponto de parada:

<https://www.w3schools.com/c/c_structs.php>

Membros da estrutura de acesso