& - devolve o endereço de memoria de uma variável

?: - operador ternário

\* - apontador para uma variável

Sizeof() – devolve o tamanho que a variável ocupa em memoria

Em c cria código para a maquina real

Para máquinas distintas é preciso re-compilar o código (ex: Windows != Linux)

Return 0 – execução com sucesso de uma função

Se fizermos return outro nr poderá simbolizar que acabou com determinado erro

Main(void) pq não recebe argumentos

Compilação – transformar o código c em código maquina

Compilação: pré-processamento, compilação, assemblagem e linkagem

Gcc é o nome do compilador (da aplicação)

W- warning

All – all

Wall – todos os avisos

Deve-se sempre resolver os avisos

-o hello serve para decidir o nome que terá o executável

-save-temps é para ficar com os ficheiros gerados durante a compilação

gcc -Wall -save-temps hello.c -o hello

hello.c -> hello.i -> hello.s -> hello.o -> printf.o

-g é para usar sempre que precisarmos de informação de debug

-c vai ser para utilizar os Makefiles

Fontes de erro:

Preprocessador – falta de ficheiros de include

Parser – erros de sintaxe

Assembler – na sintaxe do código assembly

Linker – falta de bibliotecas

Diretivas para o pré-processador começam com #

Inserir conteúdo de outro ficheiro no que é compilado: #include

Definição de macros e constantes: #define

Condicional: #if , #ifdef , #ifndef , #else , #elif , #endif

Antes da conpilação, o preprocessador(?) lê o código e transforma-o

Entre < e > - bibliotecas | ex: <stdio.h>

#include “mydefs.h” – mydefs.h é um modulo

#define MAX 100 é uma constante

#define check(x) ((x) < MAX) verifica se o x é menor que 100

Sem os parentesis À volta do x podemos ter problemas

If(check(i)) {…}

Resulta em if(((i) < 100)) {…}

Sem parêntesis faria a + o resultado do b<100 , se por acaso em ver do x tivéssemos a+b

Char – 1 byte

Short – 2 bytes

Int – 4 bytes

Long – 8 bytes

Os unsigneds têm os mesmos bytes mas não têm sinal, são só números positivos, por isso vai dar para guardar até o dobro da quantidade (mas só positivos)

Ex: para octeto do ip usaríamos um unsigned char, mas no caso do ip interiro já seria outra historia

Com virgula flutuante:

Float – 4 bytes

Double – 8 bytes

Long double – 16 bytes

A tabela ASCII é que permite interpretar os chars como letras

Char c = ‘A’

Na verdade estamos a guardar no char o valor 65 decimal, 41 hexadecimal, 101 octal

Int i = -42342

Unsigned int ui = 452452

Float pi=3.14f

Double db = 3.234245345455 -> não poderíamos guardar como float pq a precisão é demasiado grande

O espaço de armazenamento varia da arquitetura

Ex: long ocupa 4 bytes em IA32 e 8 bytes em maquinas x86-64

Void – conjunto de valores vazio, é um tipo de dados vazio que não se pode completar

Exemplo: void func(int n)

Int func(void)

Void\* ptr -> Definir um apontador que não especifica o tipo de dados para o qual aponta

Conversão implícita:

Int a=1000;

Char b=a; b=-24 , 8 bits menos significativos de a: 11101000

Explocita:

Float f=1.2f

Int d = (int)f // int=1 , trancagem

Sizeof(char) -> devolve o tamano do tipo de dados char, retorna 1

Sizeof(int) -> diz a quantidade de chars que ocupa

Sizeof(a) -> tamanho da variável a

O sizeof diz quantos chars ocupam o mesmo espaço que aquele tipo de dados, embora pareça que diz o numero de bytes

CHAR\_BIT – char ocupa 8 bits, que correspondem a 1 byte

Sizeof(int) \* CHAR\_BIT – numero de bits que ocupa o int

<limits.h> é necessário fazer include para usar o CHAR\_BIT INT\_MIN INT\_MAX …

<float.h> - FLT\_MIN FLT\_MAX FLT\_DIG …

Formatadores printf:

%d ou %i -> inteiros com sinal em base 10

%hhd – char

%hd – short

%ld - long

%u -> inteiros sem sinal em base 10

%hhu – char

%hu – short

%lu - long

%f – decimais em virgula flutuante

%E – notação cientifica (mantissa e expoente)

%c – caracter

%s – string

%p – imprimir o endereço de apontadores

Int a[10];

Int a1[] = {1, 2 , 5}

Ar[1000]={0} – 1000 inicializados a 0

S[100] – 100 sem inicialização

M[10][10] – matriz de 10x10 sem inicialização

Quando não é inicializado, serão guardados os últimos valores que foram usados nessa zona de memoria

Ex: int a[10], ao fazer a[10]=5 dá asneira

Não podemos: int v[5] e v={5, 6, 3, 5 ,1}, dá erro, seria necessário atrabuir os valores, por exemplo, através de um ciclo for

Não existe atributo length

{} definem o âmbito de validade das instruções

Tamanho do array: sizeof(a) / sizeof( ………..nao consegui ver o resto mas é importante

Sizeof(a) sendo int[] a vai imprimir sempre 9 em x86-64

String são arrays do tipo char terminadas com o caracter NUL

NUL representa-se em C através de \0

Valor 0 != caracter NUL

S[10}=”abc” seria um array com primeiros 4 elementos a, b, c e \0 e depois os restantes seriam outros números, lixo da memoria

Temos de meter o 0 para fechar a string ou entao para imprimir só o que queremos vai imprimir também lixo